US006035332A

United States Patent [19]

Ingrassia, Jr. et al.

[11] Patent Number: 6,035,332

[45] Date of Patent: Mar. 7, 2000

[54] METHOD FOR MONITORING USER INTERACTIONS WITH WEB PAGES FROM WEB SERVER USING DATA AND COMMAND LISTS FOR MAINTAINING INFORMATION VISITED AND ISSUED BY PARTICIPANTS

[75] Inventors: **Michael I. Ingrassia, Jr.**, Edison; **James A. Shelton**, Holmdel; **Thomas M. Rowland**, Fair Haven, all of N.J.

[73] Assignee: **NCR Corporation**, Dayton, Ohio

[21] Appl. No.: **08/944,759**

[22] Filed: **Oct. 6, 1997**

[51] **Int. Cl.**[7] **G06F 13/00**

[52] **U.S. Cl.** **709/224**; 709/218; 709/223

[58] **Field of Search** 395/761; 705/10; 709/224, 227, 218, 223

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,260,878	11/1993	Luppy .	
5,535,256	7/1996	Maloney et al. .	
5,544,649	8/1996	David et al. .	
5,715,453	2/1998	Stewart .	
5,737,619	4/1998	Judson	395/761
5,742,768	4/1998	Gennaro et al. .	
5,757,129	5/1998	Schafnitzel et al. .	
5,774,660	6/1998	Brendel et al. .	
5,774,670	6/1998	Montulli	395/200.57
5,784,058	7/1998	LaStrange et al. .	
5,793,972	8/1998	Shane .	
5,796,952	8/1998	Davis et al.	395/200.54
5,802,299	9/1998	Logan et al. .	
5,809,250	9/1998	Kisor .	
5,848,396	12/1998	Gerace	705/10

Primary Examiner—Le Hien Luu

Attorney, Agent, or Firm—Ying Tuo; Kenneth M. Berner

[57] **ABSTRACT**

Described is a mechanism for dependably tracking web page activities among a group of browsers. The web browsers retrieve web pages from an HTTP server, with each of the web pages embedding an applet. In response to web page activities (such as loading or unloading of a web page) performed at a browser, the respective applet reports the activities (together with the URL of the web page) to a synchronization server, which in turn stores them in a database.

37 Claims, 22 Drawing Sheets

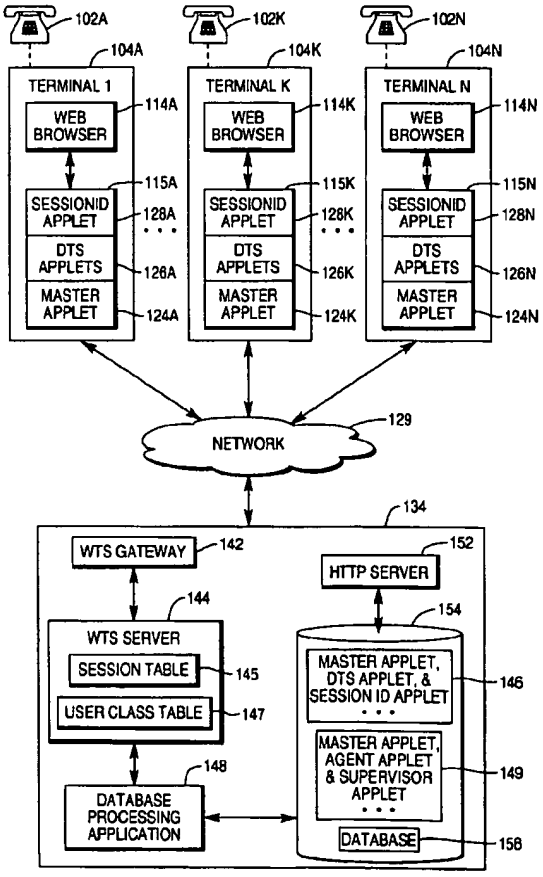
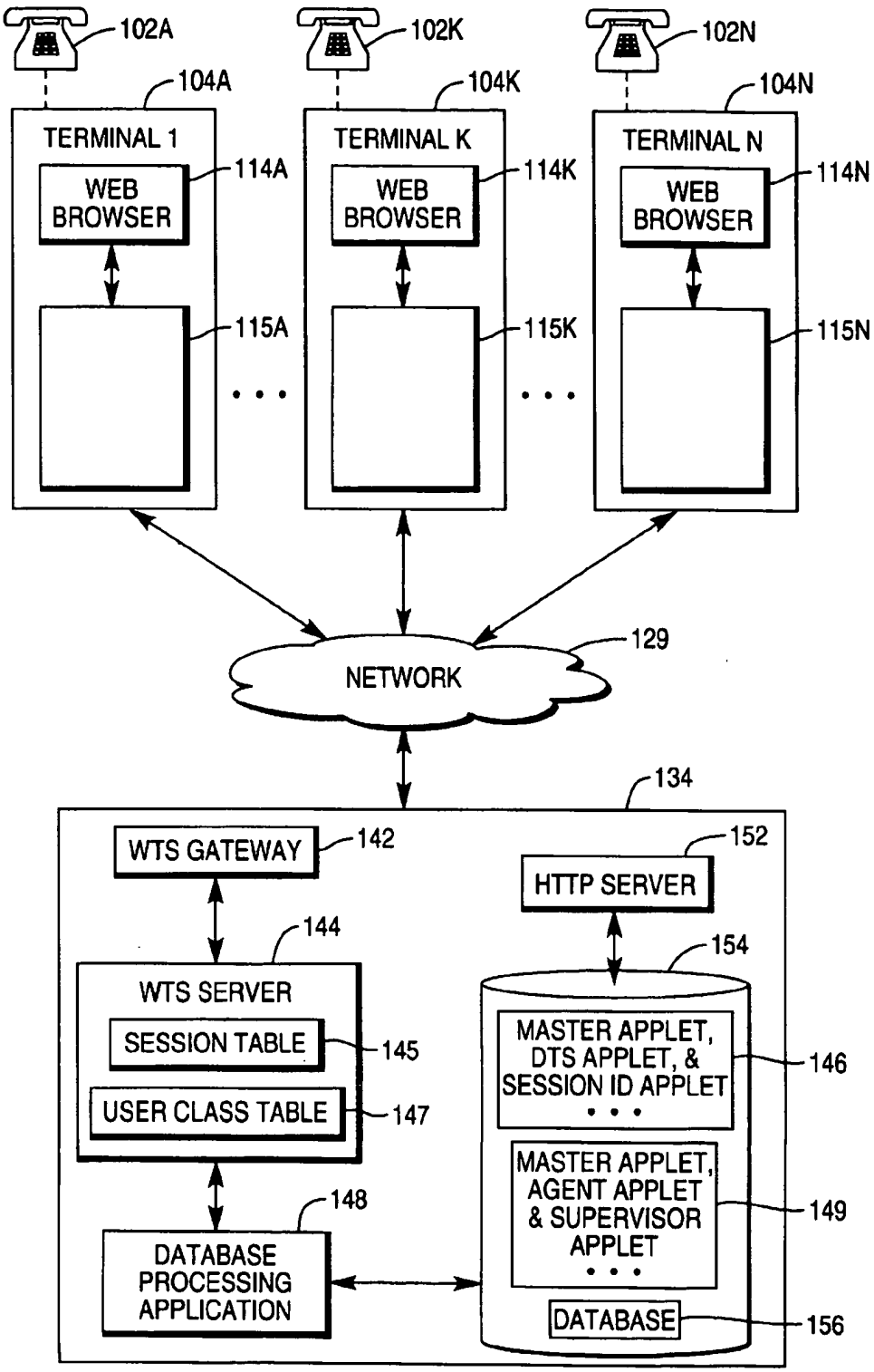


FIG. 1


102A
102K
102N
104A
104K
104N
TERMINAL 1
TERMINAL K
TERMINAL N
WEB BROWSER
WEB BROWSER
WEB BROWSER
114A
114K
114N
115A
115K
115N
129
NETWORK
134
WTS GATEWAY
142
HTTP SERVER
152
144
154
WTS SERVER
SESSION TABLE
145
USER CLASS TABLE
147
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

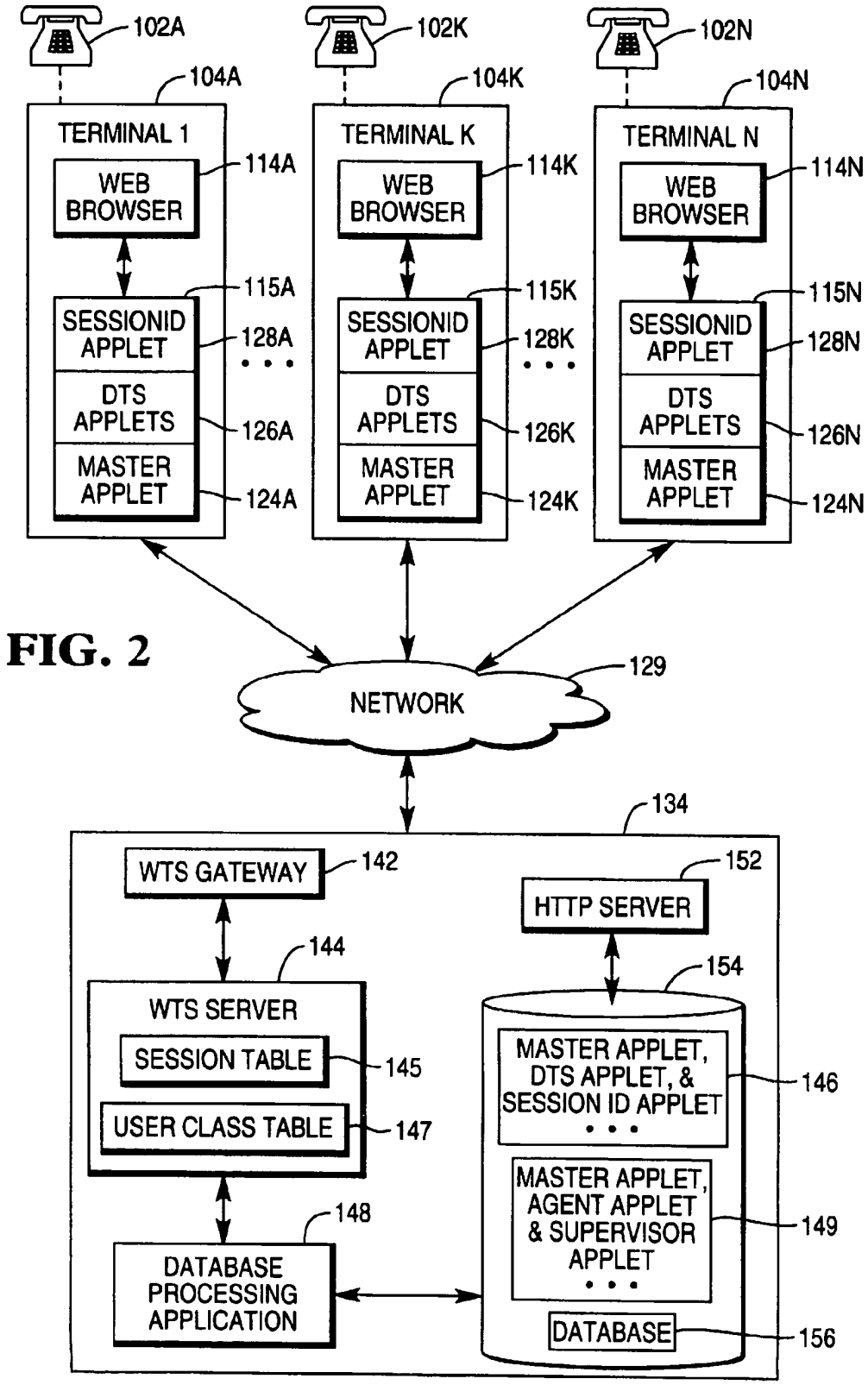

102A
102K
102N
104A
104K
104N
TERMINAL 1
TERMINAL K
TERMINAL N
WEB BROWSER
114A
114K
114N
SESSIONID APPLET
115A
115K
115N
128A
128K
128N
DTS APPLETS
126A
126K
126N
MASTER APPLET
124A
124K
124N
NETWORK
129
134
WTS GATEWAY
142
HTTP SERVER
152
144
154
WTS SERVER
SESSION TABLE
145
USER CLASS TABLE
147
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

FIG. 2

FIG. 3

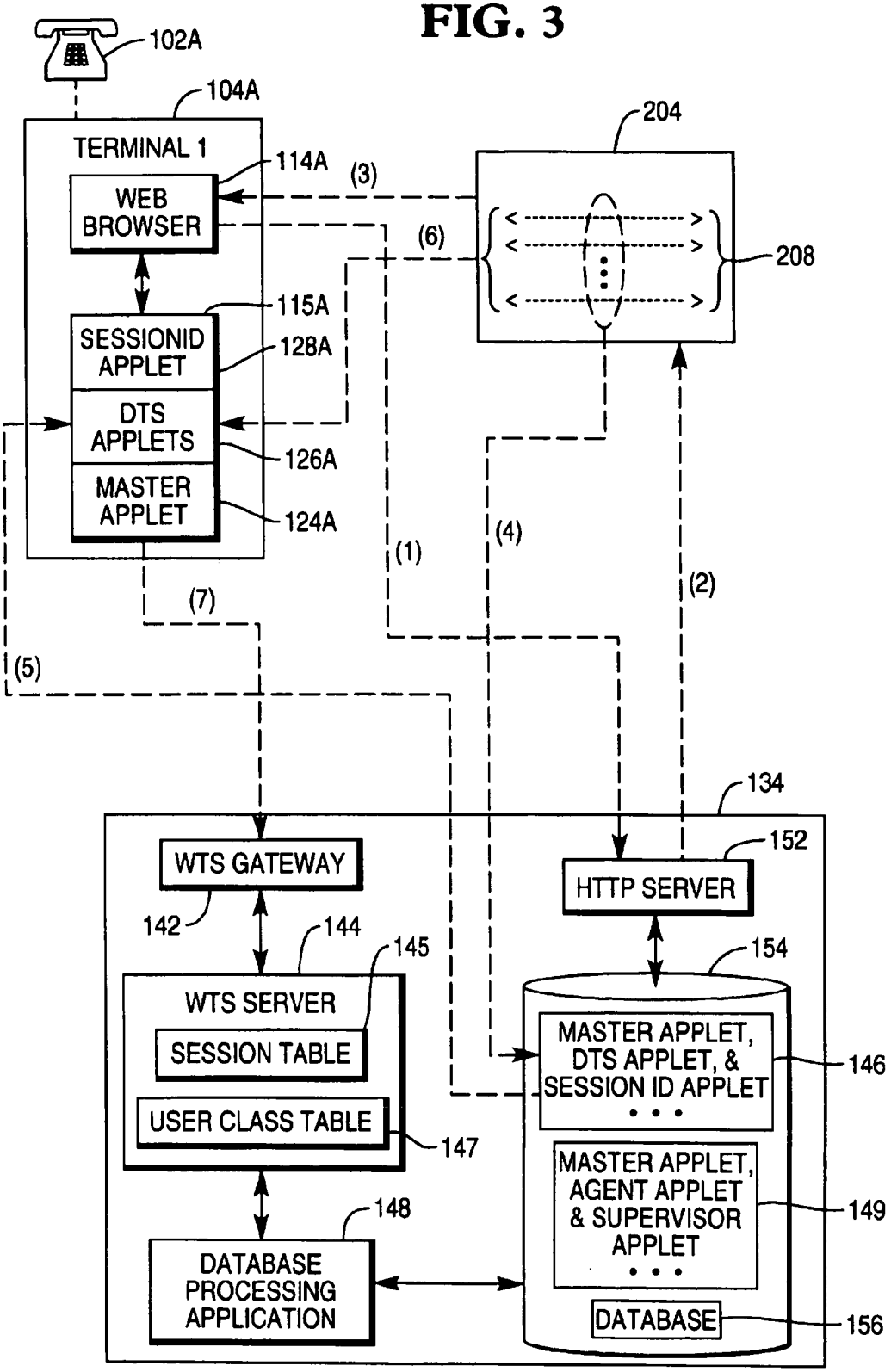

102A
104A
TERMINAL 1
114A
WEB BROWSER
(3)
204
(6)
208
115A
SESSIONID APPLET
128A
DTS APPLETS
126A
MASTER APPLET
124A
(1)
(4)
(2)
(7)
(5)
134
WTS GATEWAY
142
144
145
WTS SERVER
SESSION TABLE
USER CLASS TABLE
147
148
DATABASE PROCESSING APPLICATION
152
HTTP SERVER
154
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
DATABASE
156

FIG. 4

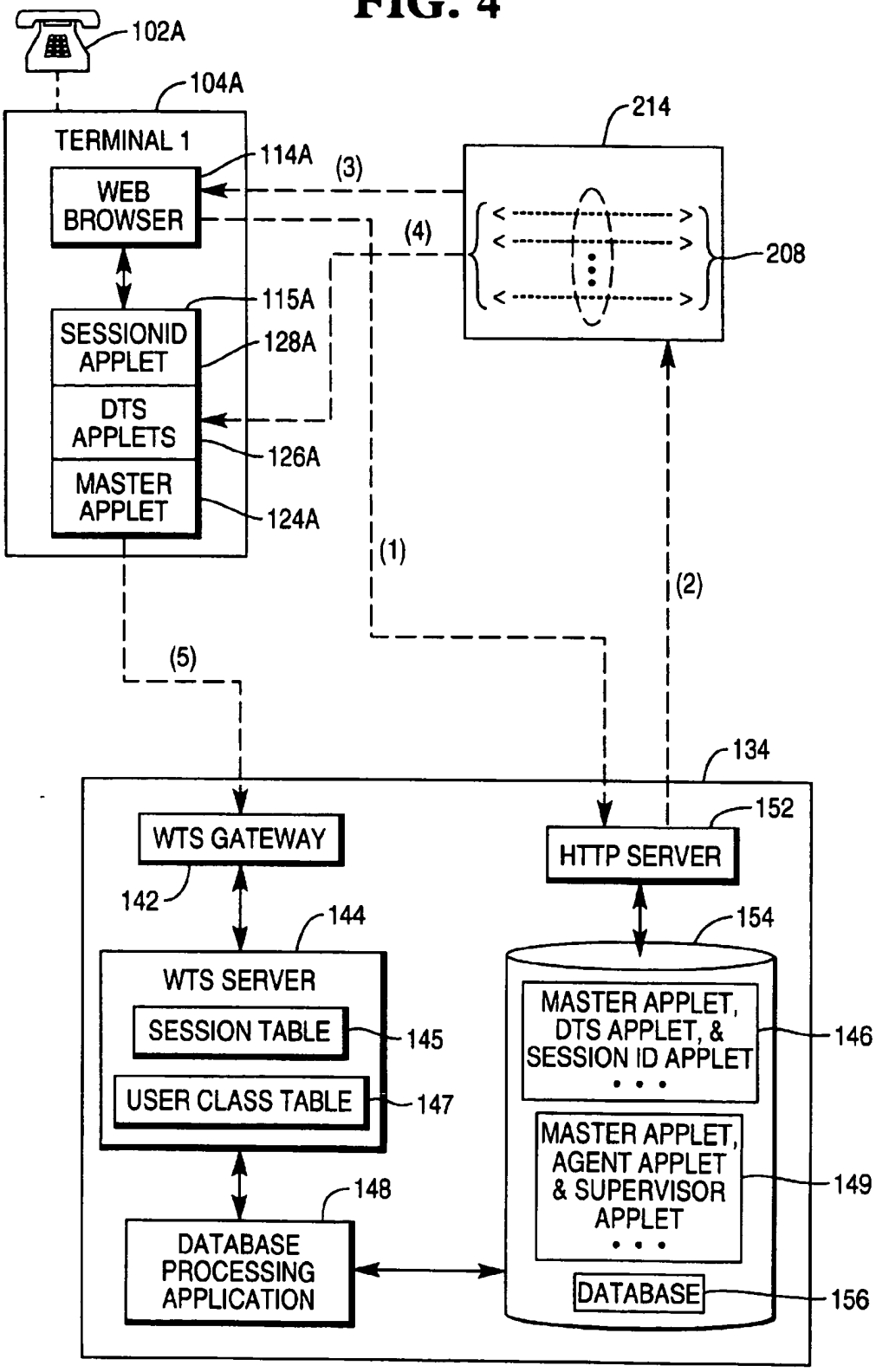

102A
104A
TERMINAL 1
114A
WEB BROWSER
(3)
214
(4)
208
115A
SESSIONID APPLET
128A
DTS APPLETS
126A
MASTER APPLET
124A
(1)
(2)
(5)
134
WTS GATEWAY
142
152
HTTP SERVER
144
154
WTS SERVER
SESSION TABLE
145
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
USER CLASS TABLE
147
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

FIG. 5

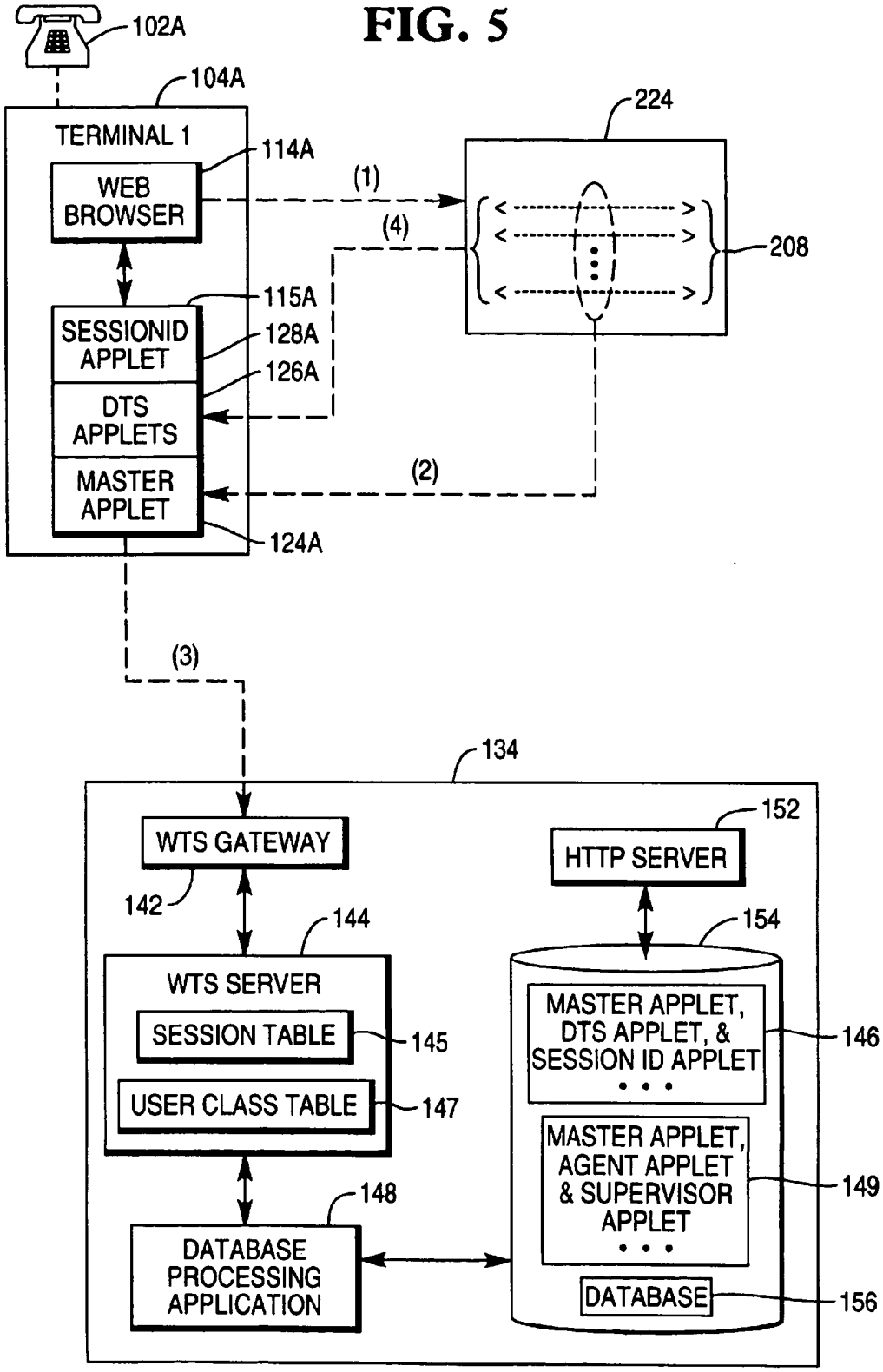

102A
104A
TERMINAL 1
114A
WEB BROWSER
(1)
224
(4)
208
115A
SESSIONID APPLET
128A
126A
DTS APPLETS
MASTER APPLET
(2)
124A
(3)
134
WTS GATEWAY
142
152
HTTP SERVER
144
154
WTS SERVER
SESSION TABLE
145
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
USER CLASS TABLE
147
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

FIG. 6

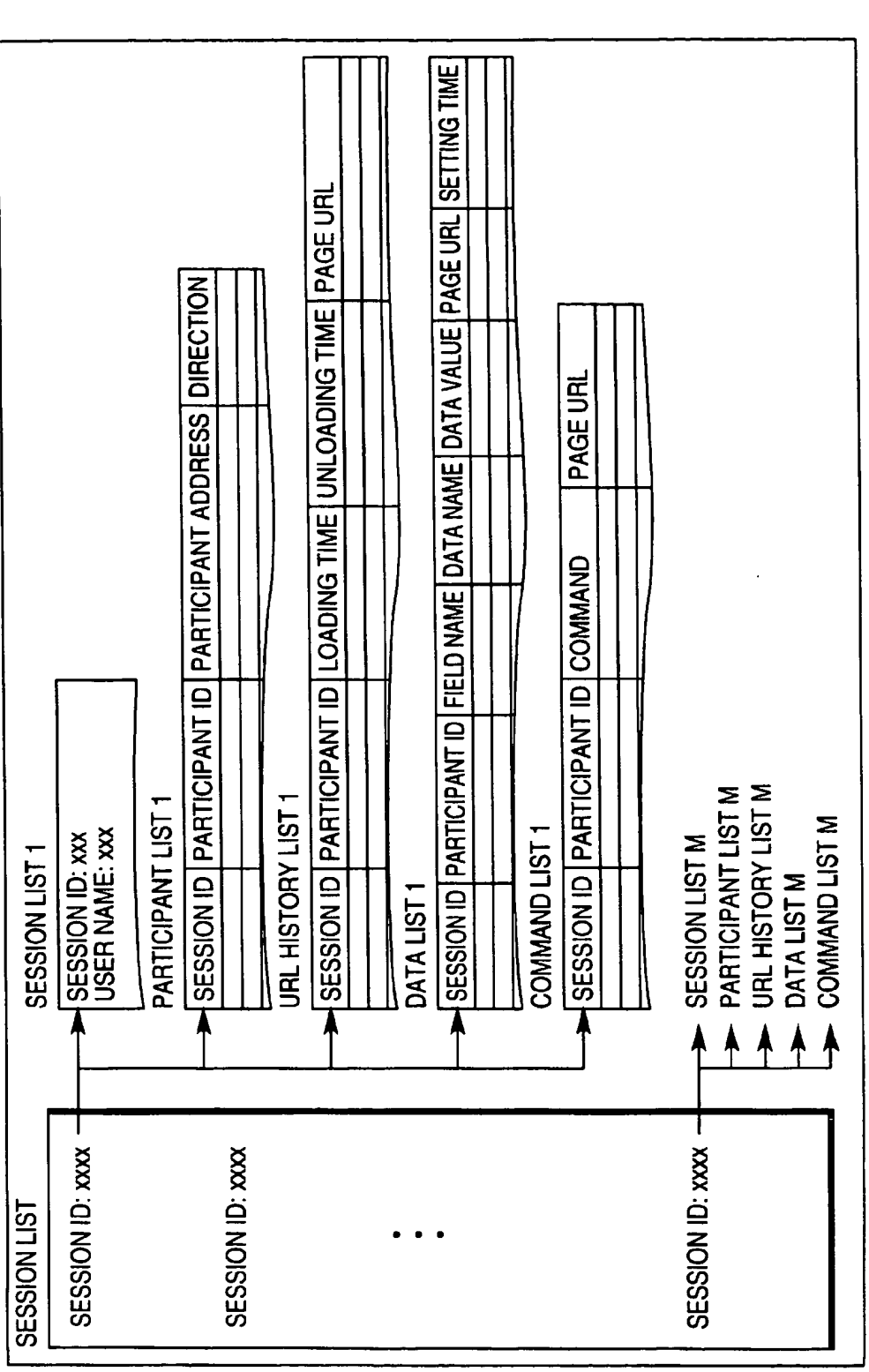
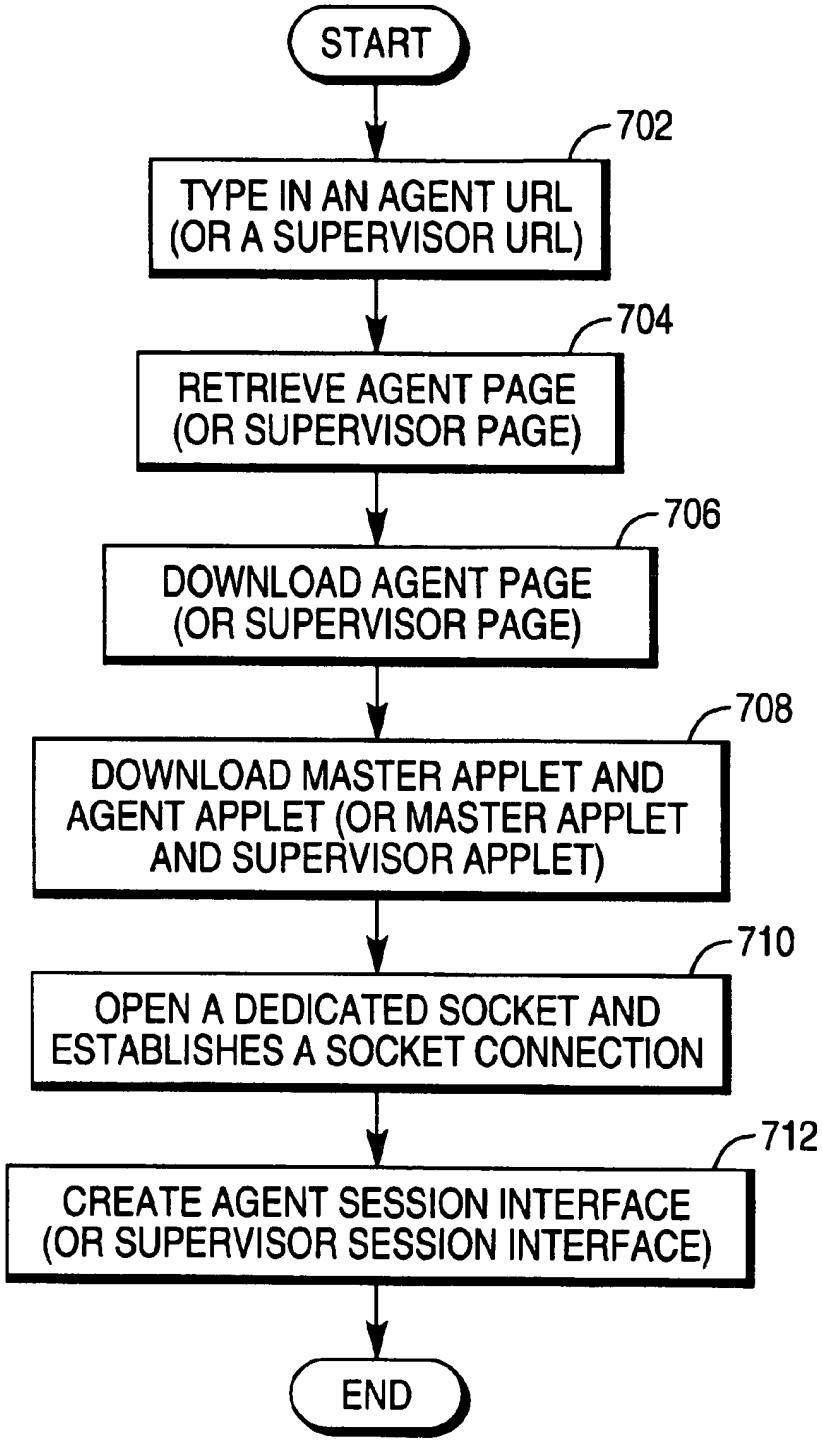

145
SESSION LIST
SESSION ID: xxxx
SESSION ID: xxxx
SESSION ID: xxxx
SESSION LIST 1
SESSION ID: xxx
USER NAME: xxx
PARTICIPANT LIST 1
SESSION ID
PARTICIPANT ID
PARTICIPANT ADDRESS
DIRECTION
URL HISTORY LIST 1
SESSION ID
PARTICIPANT ID
LOADING TIME
UNLOADING TIME
PAGE URL
DATA LIST 1
SESSION ID
PARTICIPANT ID
FIELD NAME
DATA NAME
DATA VALUE
PAGE URL
SETTING TIME
COMMAND LIST 1
SESSION ID
PARTICIPANT ID
COMMAND
PAGE URL
SESSION LIST M
PARTICIPANT LIST M
URL HISTORY LIST M
DATA LIST M
COMMAND LIST M

FIG 7


START
702
TYPE IN AN AGENT URL (OR A SUPERVISOR URL)
704
RETRIEVE AGENT PAGE (OR SUPERVISOR PAGE)
706
DOWNLOAD AGENT PAGE (OR SUPERVISOR PAGE)
708
DOWNLOAD MASTER APPLET AND AGENT APPLET (OR MASTER APPLET AND SUPERVISOR APPLET)
710
OPEN A DEDICATED SOCKET AND ESTABLISHES A SOCKET CONNECTION
712
CREATE AGENT SESSION INTERFACE (OR SUPERVISOR SESSION INTERFACE)
END

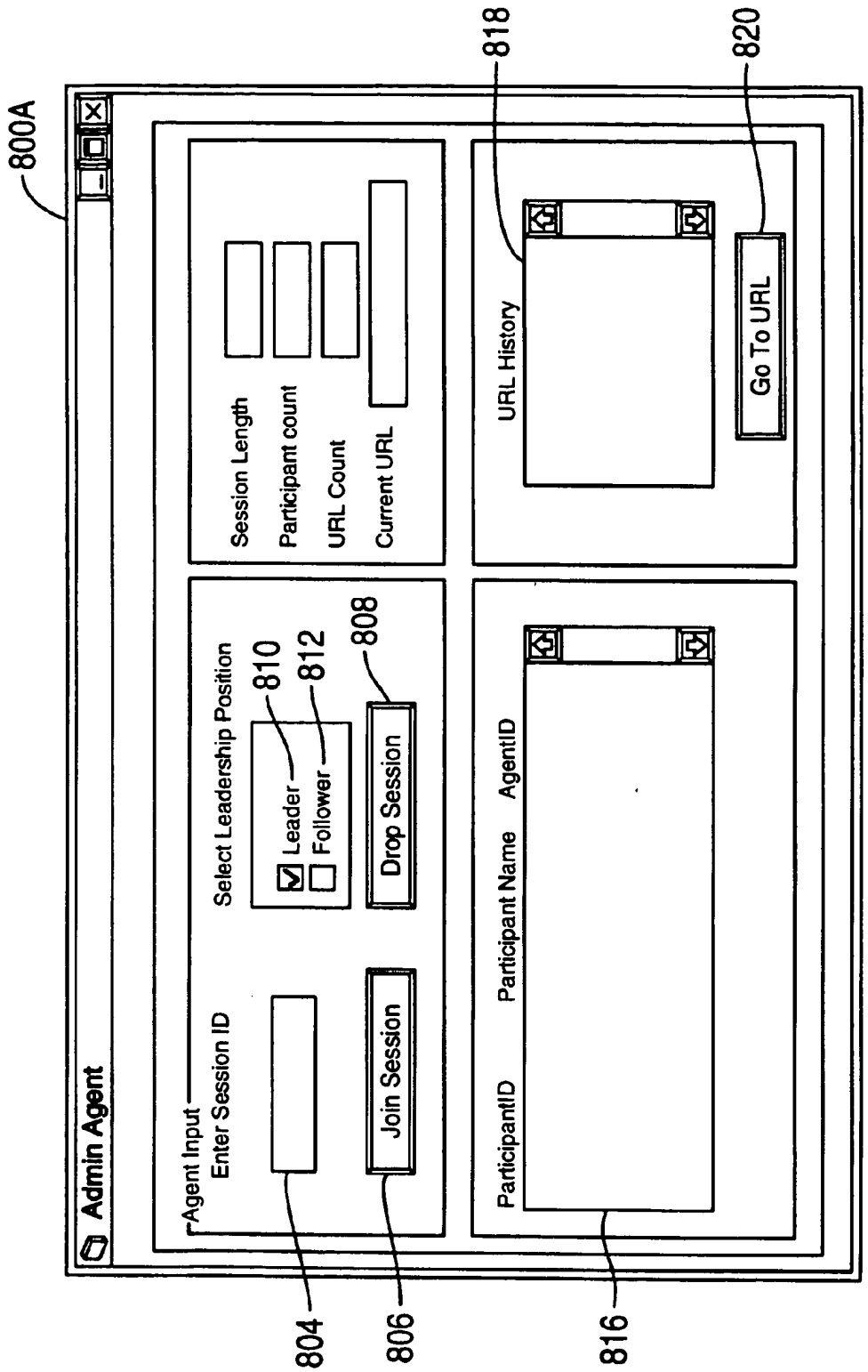
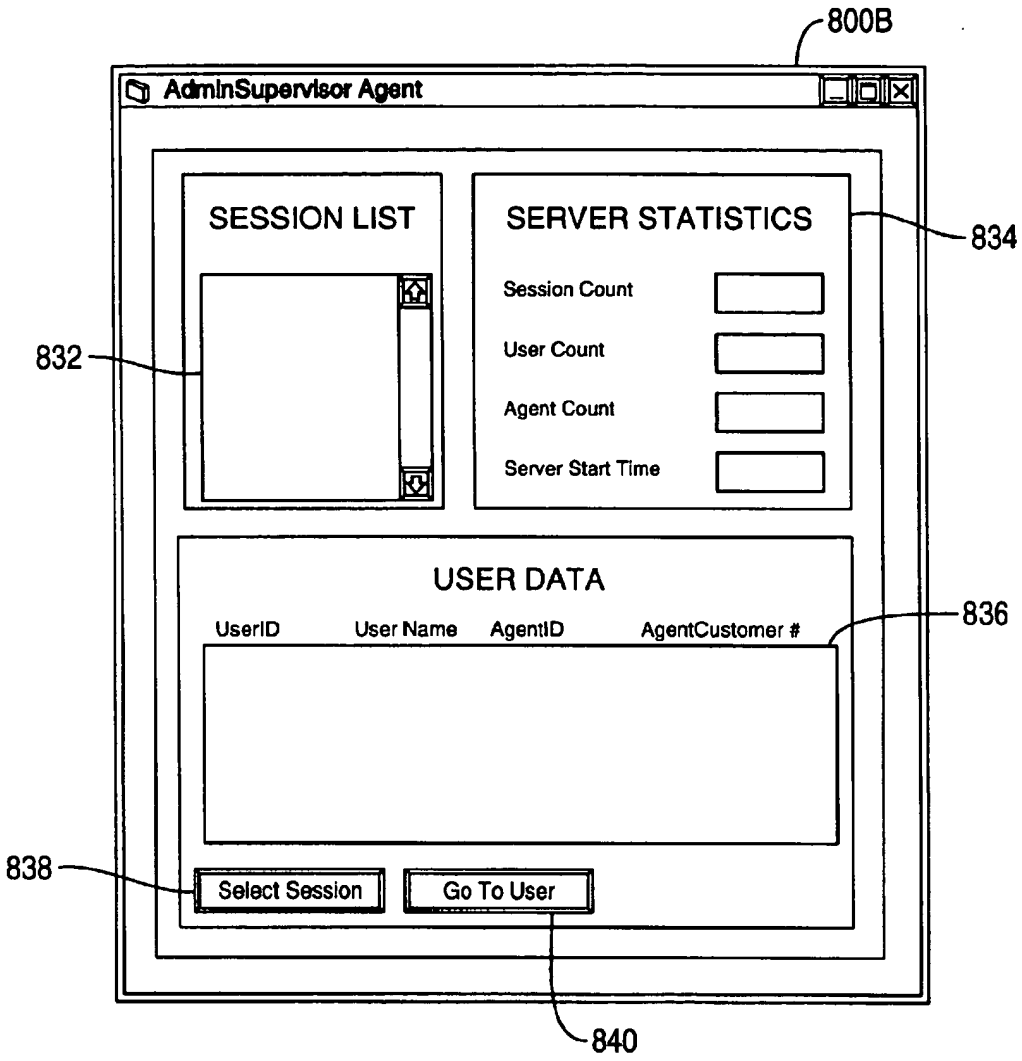

800A
Admin Agent
Agent Input
Enter Session ID
Join Session
Select Leadership Position
Leader
Follower
Drop Session
Session Length
Participant count
URL Count
Current URL
URL History
Go To URL
ParticipantID
Participant Name
AgentID
804
806
808
810
812
816
818
820

FIG. 8A

FIG. 8B


800B
AdminSupervisor Agent
SESSION LIST
SERVER STATISTICS
Session Count
User Count
Agent Count
Server Start Time
832
834
USER DATA
UserID
User Name
AgentID
AgentCustomer #
836
838
Select Session
Go To User
840

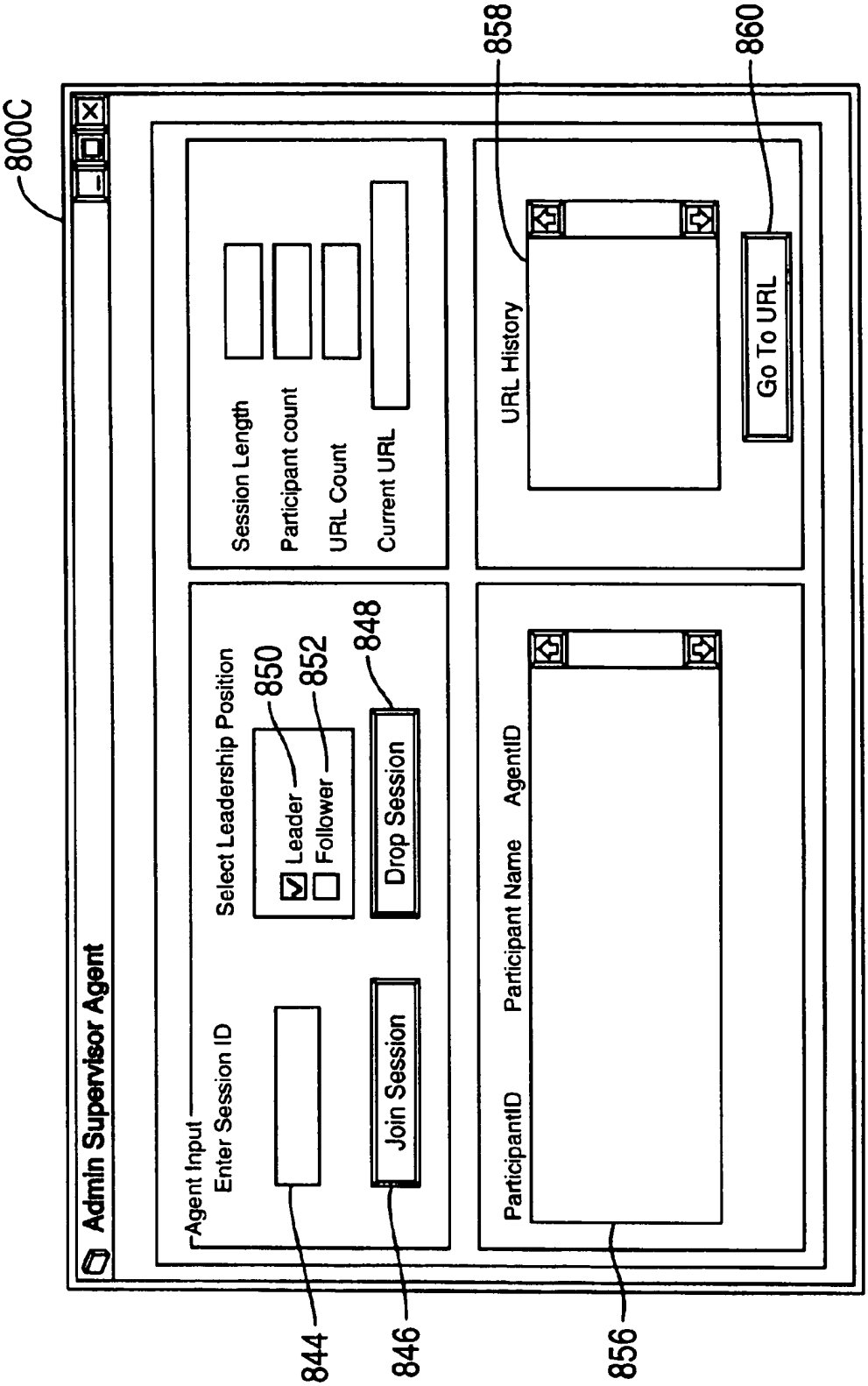
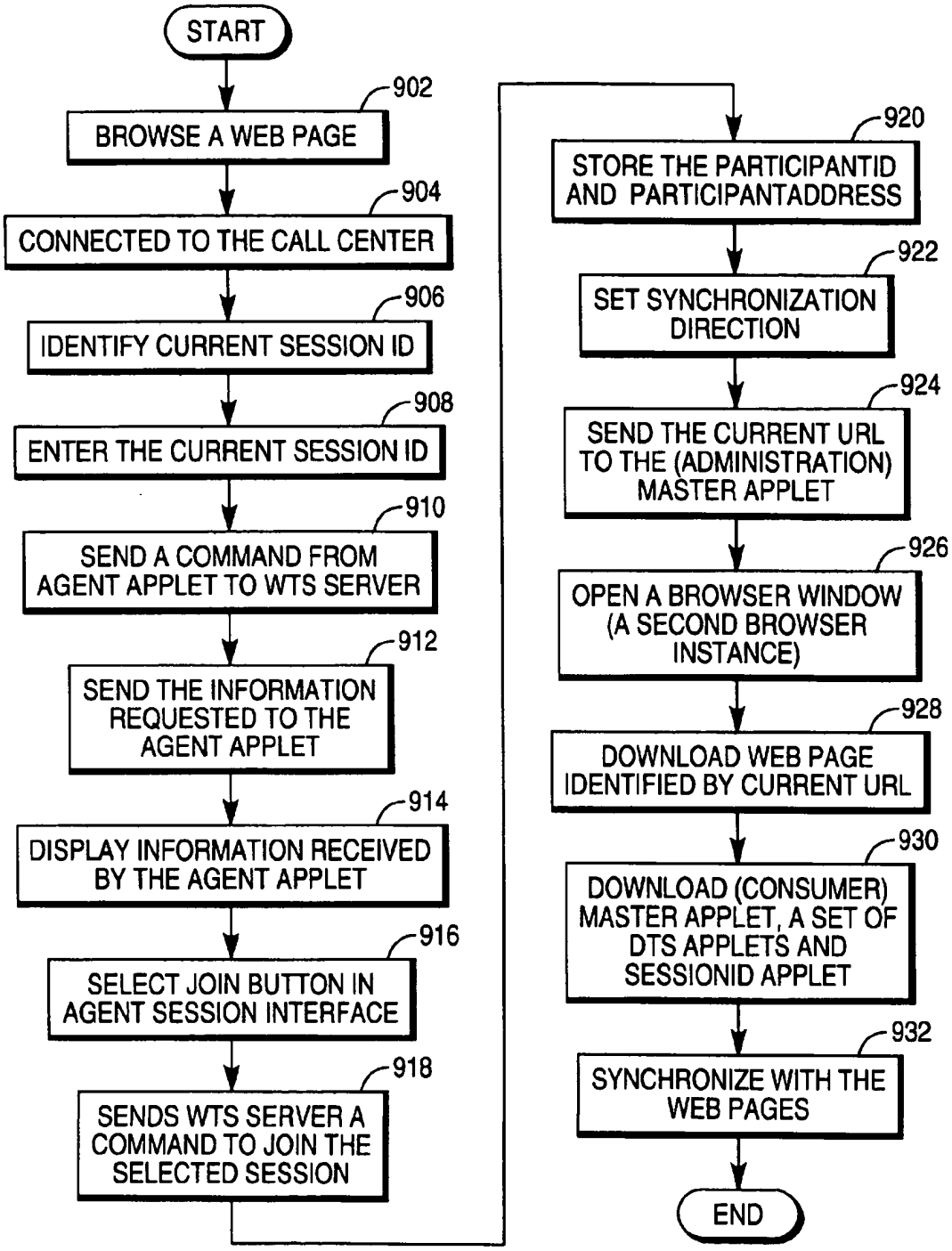

800C
858
860
Admin Supervisor Agent
Session Length
Participant count
URL Count
Current URL
URL History
Go To URL
Select Leadership Position
850
852
848
Leader
Follower
Drop Session
Agent Input
Enter Session ID
Join Session
ParticipantID
Participant Name
AgentID
844
846
856

FIG. 8C

FIG. 9


START
902
BROWSE A WEB PAGE
904
CONNECTED TO THE CALL CENTER
906
IDENTIFY CURRENT SESSION ID
908
ENTER THE CURRENT SESSION ID
910
SEND A COMMAND FROM AGENT APPLET TO WTS SERVER
912
SEND THE INFORMATION REQUESTED TO THE AGENT APPLET
914
DISPLAY INFORMATION RECEIVED BY THE AGENT APPLET
916
SELECT JOIN BUTTON IN AGENT SESSION INTERFACE
918
SENDS WTS SERVER A COMMAND TO JOIN THE SELECTED SESSION
920
STORE THE PARTICIPANTID AND PARTICIPANTADDRESS
922
SET SYNCHRONIZATION DIRECTION
924
SEND THE CURRENT URL TO THE (ADMINISTRATION) MASTER APPLET
926
OPEN A BROWSER WINDOW (A SECOND BROWSER INSTANCE)
928
DOWNLOAD WEB PAGE IDENTIFIED BY CURRENT URL
930
DOWNLOAD (CONSUMER) MASTER APPLET, A SET OF DTS APPLETS AND SESSIONID APPLET
932
SYNCHRONIZE WITH THE WEB PAGES
END

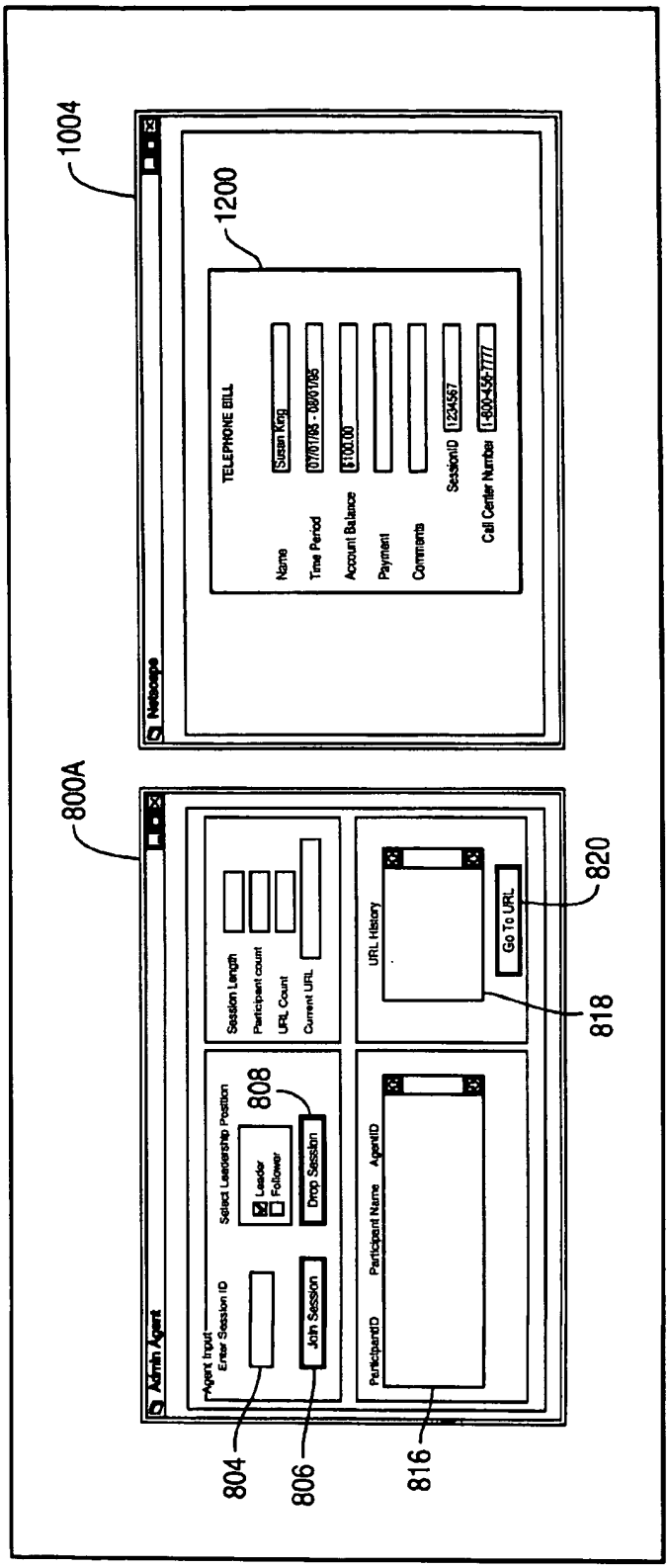

1004
1200
Netscape
TELEPHONE BILL
Name
Time Period
Account Balance
Payment
Comments
SessionID
Call Center Number
Susan King
07/01/98 - 08/01/98
$100.00
1234567
1-800-555-7777
800A
Admin Agent
Session Length
Participant count
URL Count
Current URL
Select Leadership Position
Leader
Follower
Drop Session
808
Agent Input
Enter Session ID
Join Session
URL History
Go To URL
820
818
ParticipantID
Participant Name
AgentID
804
806
816

FIG. 10

FIG. 11

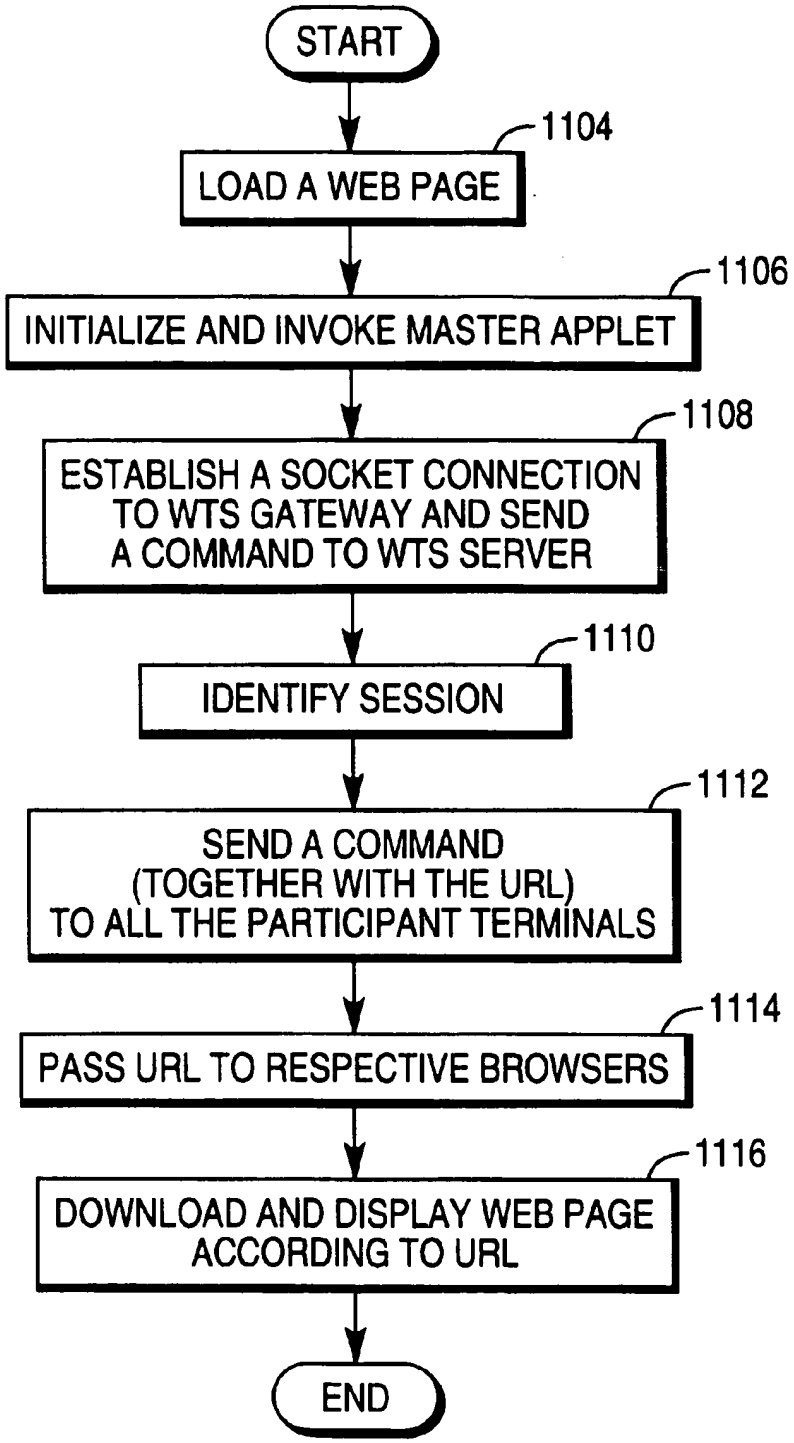

START
1104
LOAD A WEB PAGE
1106
INITIALIZE AND INVOKE MASTER APPLET
1108
ESTABLISH A SOCKET CONNECTION TO WTS GATEWAY AND SEND A COMMAND TO WTS SERVER
1110
IDENTIFY SESSION
1112
SEND A COMMAND (TOGETHER WITH THE URL) TO ALL THE PARTICIPANT TERMINALS
1114
PASS URL TO RESPECTIVE BROWSERS
1116
DOWNLOAD AND DISPLAY WEB PAGE ACCORDING TO URL
END

FIG. 12A

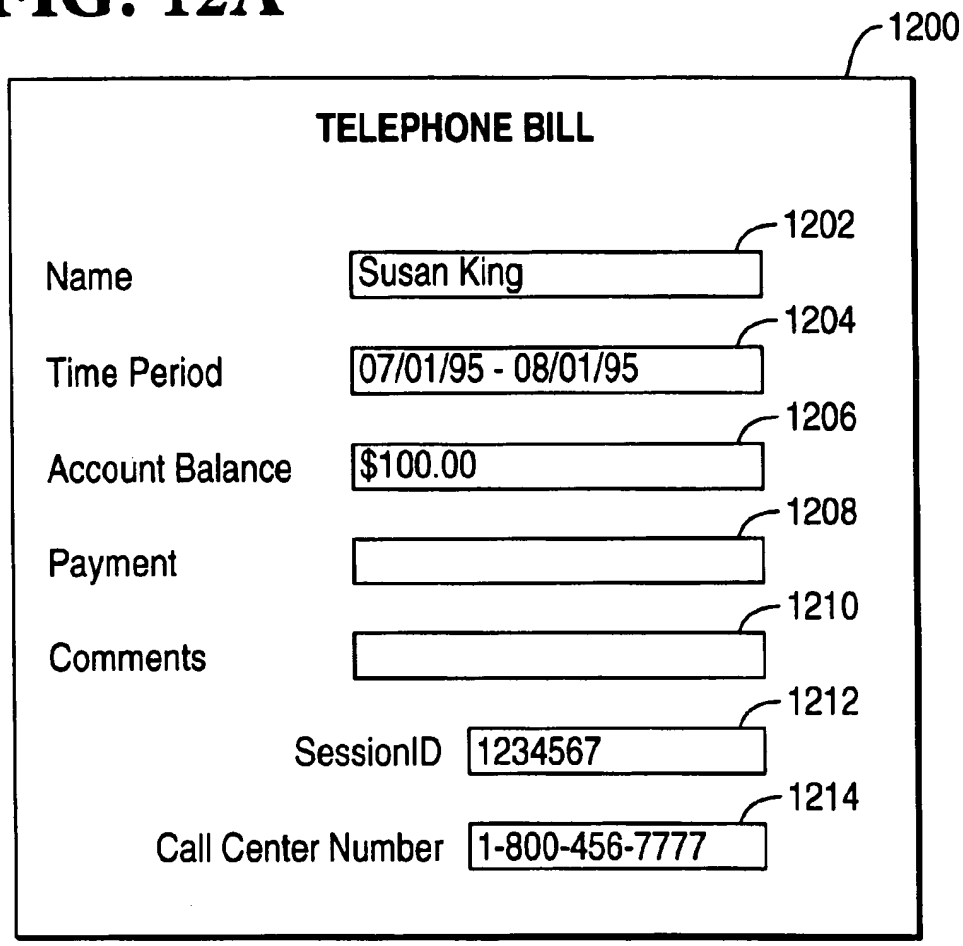

1200
TELEPHONE BILL
1202
Name
Susan King
1204
Time Period
07/01/95 - 08/01/95
1206
Account Balance
$100.00
1208
Payment
1210
Comments
1212
SessionID
1234567
1214
Call Center Number
1-800-456-7777

FIG. 12B

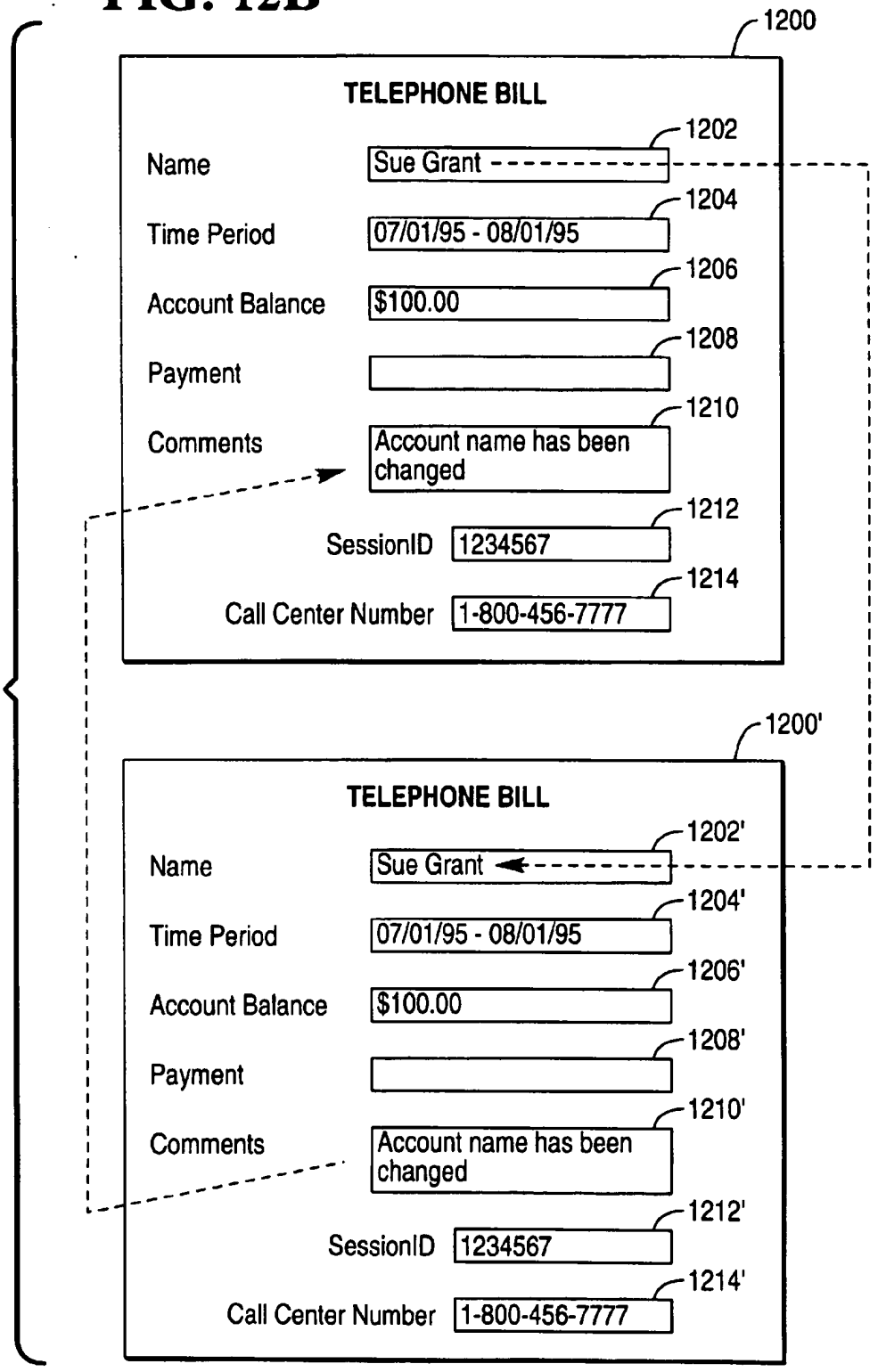

1200
TELEPHONE BILL
1202
Name
Sue Grant
1204
Time Period
07/01/95 - 08/01/95
1206
Account Balance
$100.00
1208
Payment
1210
Comments
Account name has been changed
1212
SessionID
1234567
1214
Call Center Number
1-800-456-7777
1200'
TELEPHONE BILL
1202'
Name
Sue Grant
1204'
Time Period
07/01/95 - 08/01/95
1206'
Account Balance
$100.00
1208'
Payment
1210'
Comments
Account name has been changed
1212'
SessionID
1234567
1214'
Call Center Number
1-800-456-7777

FIG 13

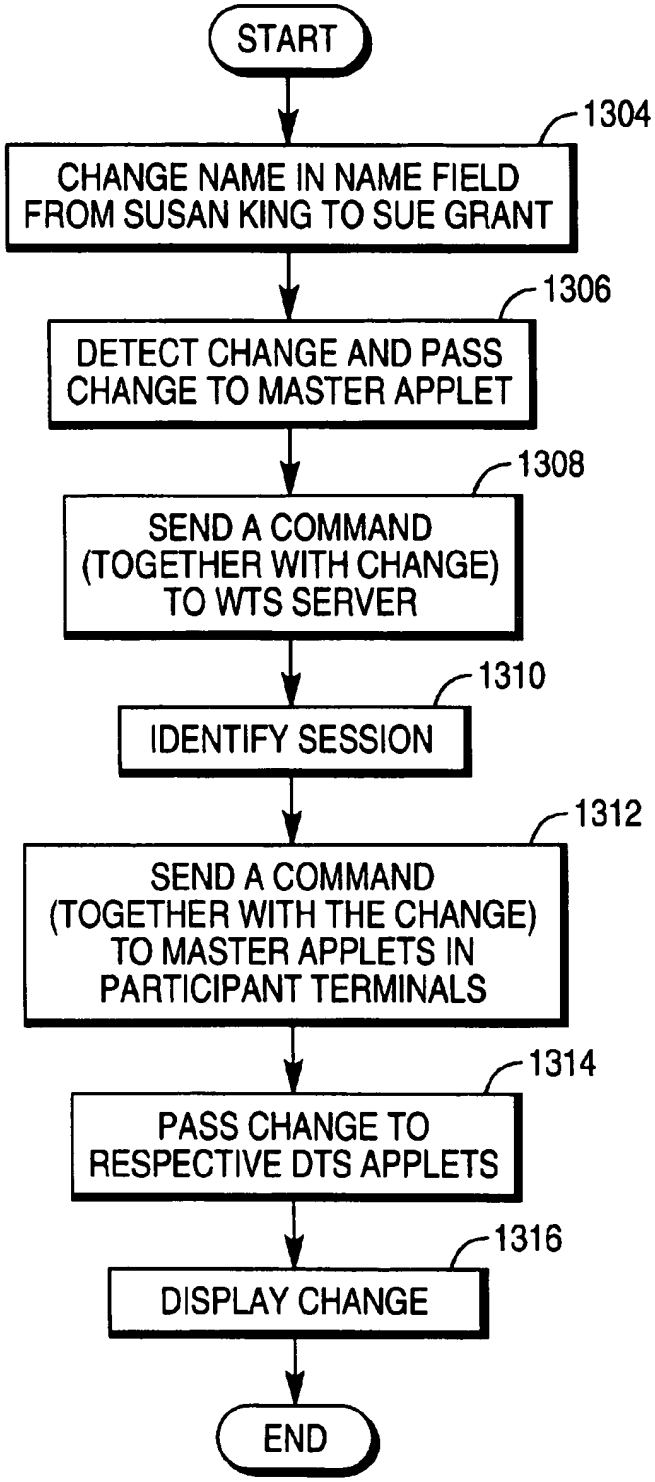

START
1304
CHANGE NAME IN NAME FIELD FROM SUSAN KING TO SUE GRANT
1306
DETECT CHANGE AND PASS CHANGE TO MASTER APPLET
1308
SEND A COMMAND (TOGETHER WITH CHANGE) TO WTS SERVER
1310
IDENTIFY SESSION
1312
SEND A COMMAND (TOGETHER WITH THE CHANGE) TO MASTER APPLETS IN PARTICIPANT TERMINALS
1314
PASS CHANGE TO RESPECTIVE DTS APPLETS
1316
DISPLAY CHANGE
END

FIG. 14

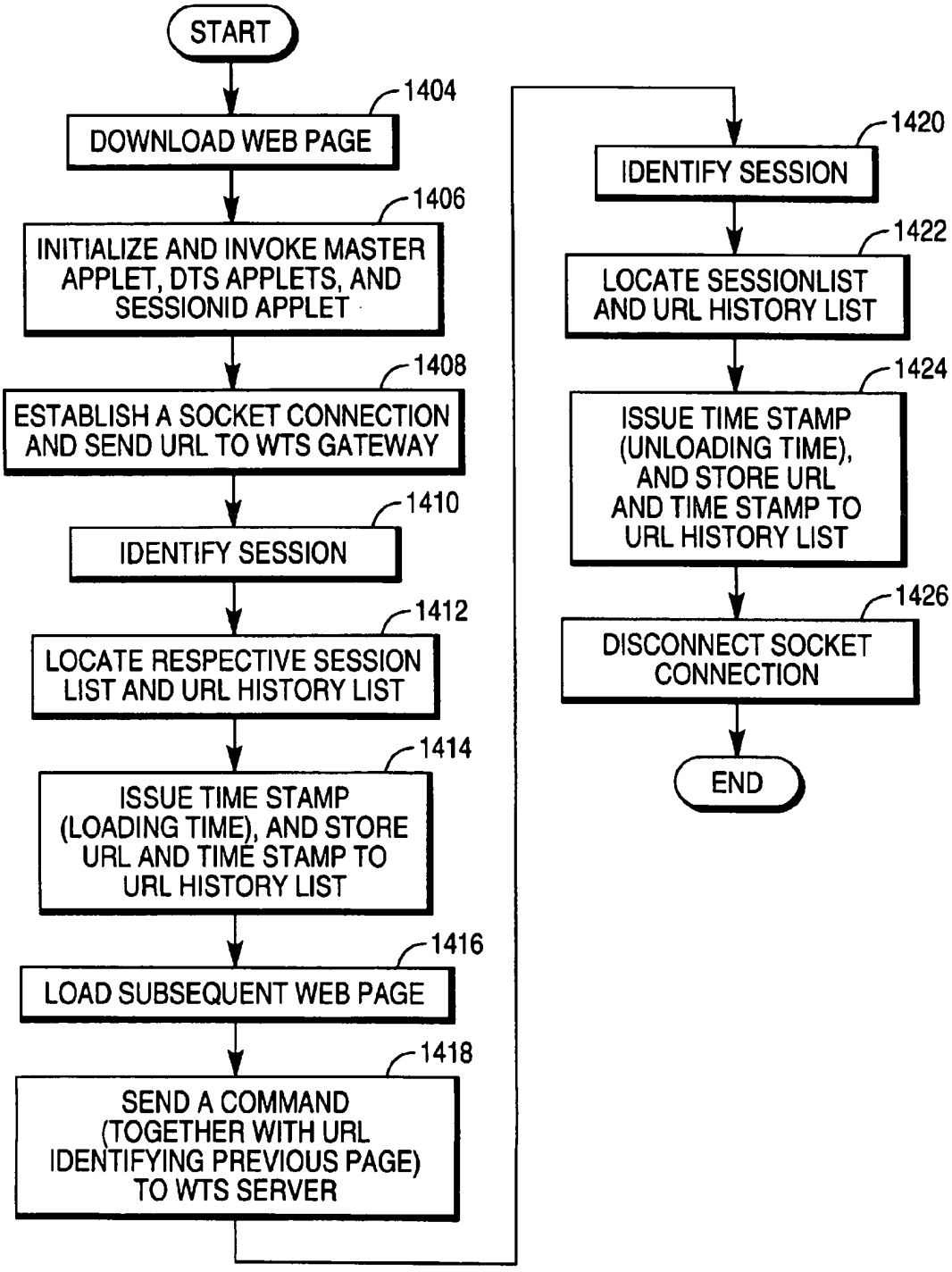

START
1404
DOWNLOAD WEB PAGE
1406
INITIALIZE AND INVOKE MASTER APPLET, DTS APPLETS, AND SESSIONID APPLET
1408
ESTABLISH A SOCKET CONNECTION AND SEND URL TO WTS GATEWAY
1410
IDENTIFY SESSION
1412
LOCATE RESPECTIVE SESSION LIST AND URL HISTORY LIST
1414
ISSUE TIME STAMP (LOADING TIME), AND STORE URL AND TIME STAMP TO URL HISTORY LIST
1416
LOAD SUBSEQUENT WEB PAGE
1418
SEND A COMMAND (TOGETHER WITH URL IDENTIFYING PREVIOUS PAGE) TO WTS SERVER
1420
IDENTIFY SESSION
1422
LOCATE SESSIONLIST AND URL HISTORY LIST
1424
ISSUE TIME STAMP (UNLOADING TIME), AND STORE URL AND TIME STAMP TO URL HISTORY LIST
1426
DISCONNECT SOCKET CONNECTION
END

FIG 15

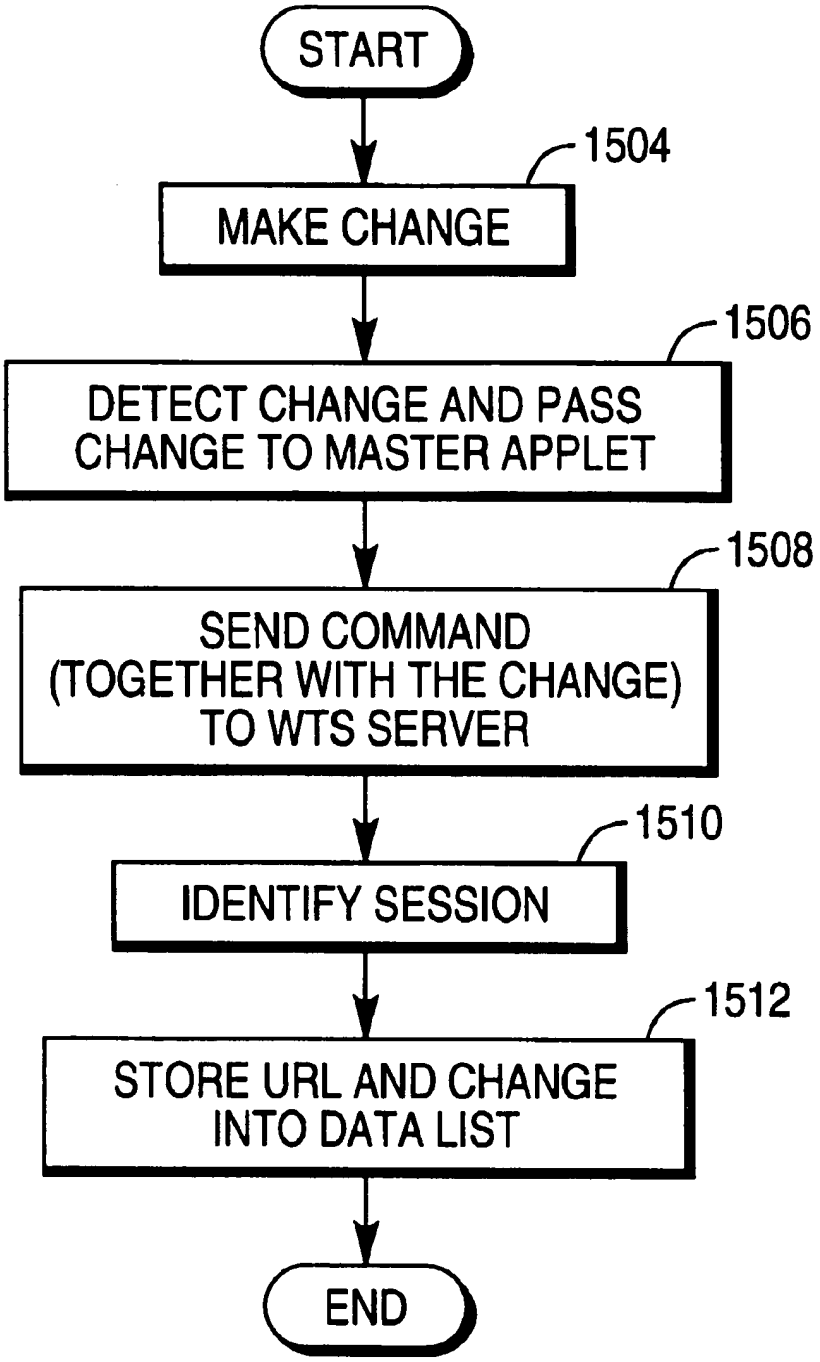
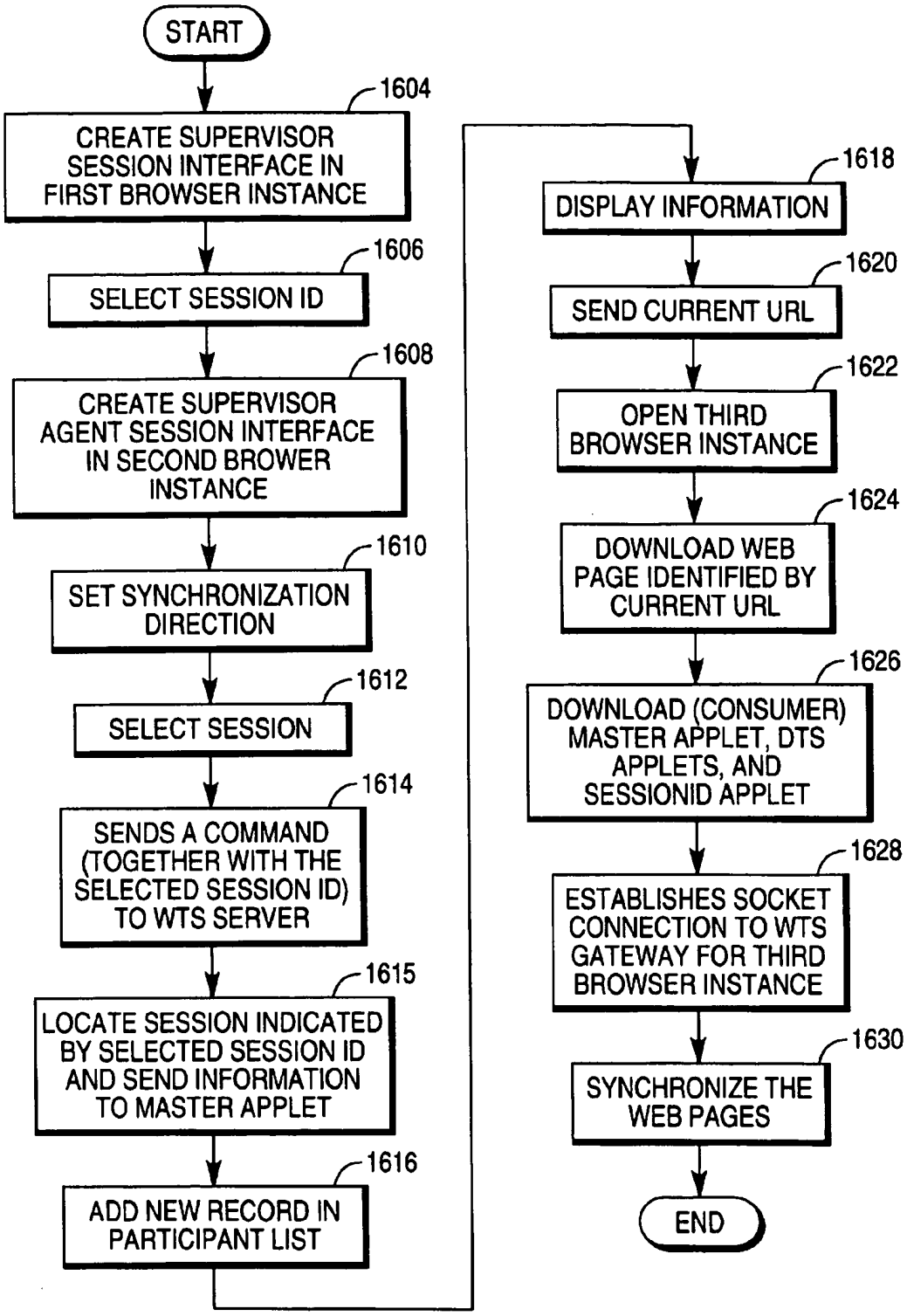

START
1504
MAKE CHANGE
1506
DETECT CHANGE AND PASS CHANGE TO MASTER APPLET
1508
SEND COMMAND (TOGETHER WITH THE CHANGE) TO WTS SERVER
1510
IDENTIFY SESSION
1512
STORE URL AND CHANGE INTO DATA LIST
END

FIG. 16


START
1604
CREATE SUPERVISOR SESSION INTERFACE IN FIRST BROWSER INSTANCE
1606
SELECT SESSION ID
1608
CREATE SUPERVISOR AGENT SESSION INTERFACE IN SECOND BROWER INSTANCE
1610
SET SYNCHRONIZATION DIRECTION
1612
SELECT SESSION
1614
SENDS A COMMAND (TOGETHER WITH THE SELECTED SESSION ID) TO WTS SERVER
1615
LOCATE SESSION INDICATED BY SELECTED SESSION ID AND SEND INFORMATION TO MASTER APPLET
1616
ADD NEW RECORD IN PARTICIPANT LIST
1618
DISPLAY INFORMATION
1620
SEND CURRENT URL
1622
OPEN THIRD BROWSER INSTANCE
1624
DOWNLOAD WEB PAGE IDENTIFIED BY CURRENT URL
1626
DOWNLOAD (CONSUMER) MASTER APPLET, DTS APPLETS, AND SESSIONID APPLET
1628
ESTABLISHES SOCKET CONNECTION TO WTS GATEWAY FOR THIRD BROWSER INSTANCE
1630
SYNCHRONIZE THE WEB PAGES
END

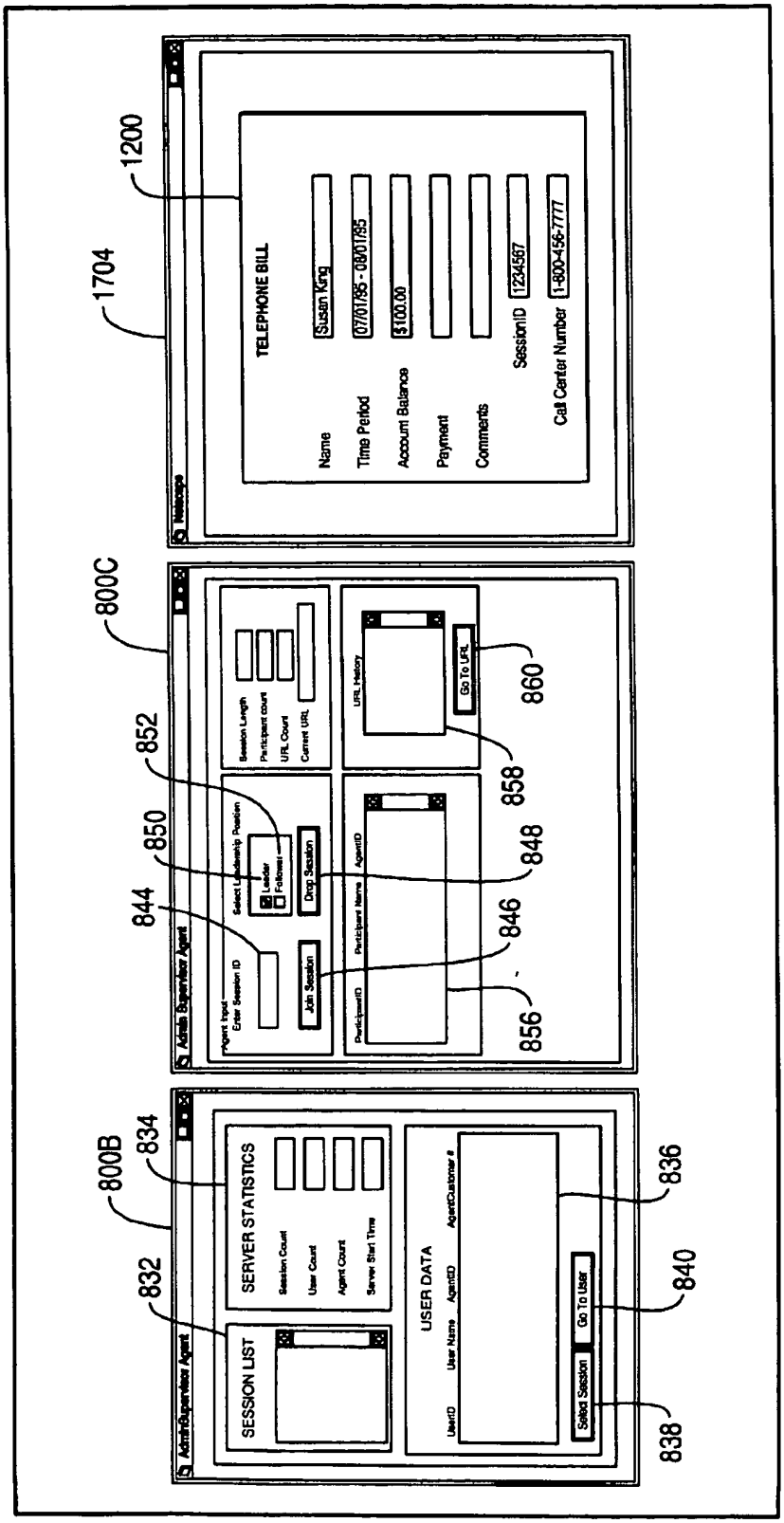

1200
1704
TELEPHONE BILL
Name
Susan King
Time Period
07/01/85 - 08/01/85
Account Balance
$100.00
Payment
Comments
SessionID
1234567
Call Center Number
1-800-456-7777
800C
852
850
844
846
848
858
860
856
Join Session
Drop Session
Leader
Follower
Go To URL
800B
834
832
836
840
838
SERVER STATISTICS
SESSION LIST
USER DATA
Select Session
Go To User

FIG. 17

FIG. 18

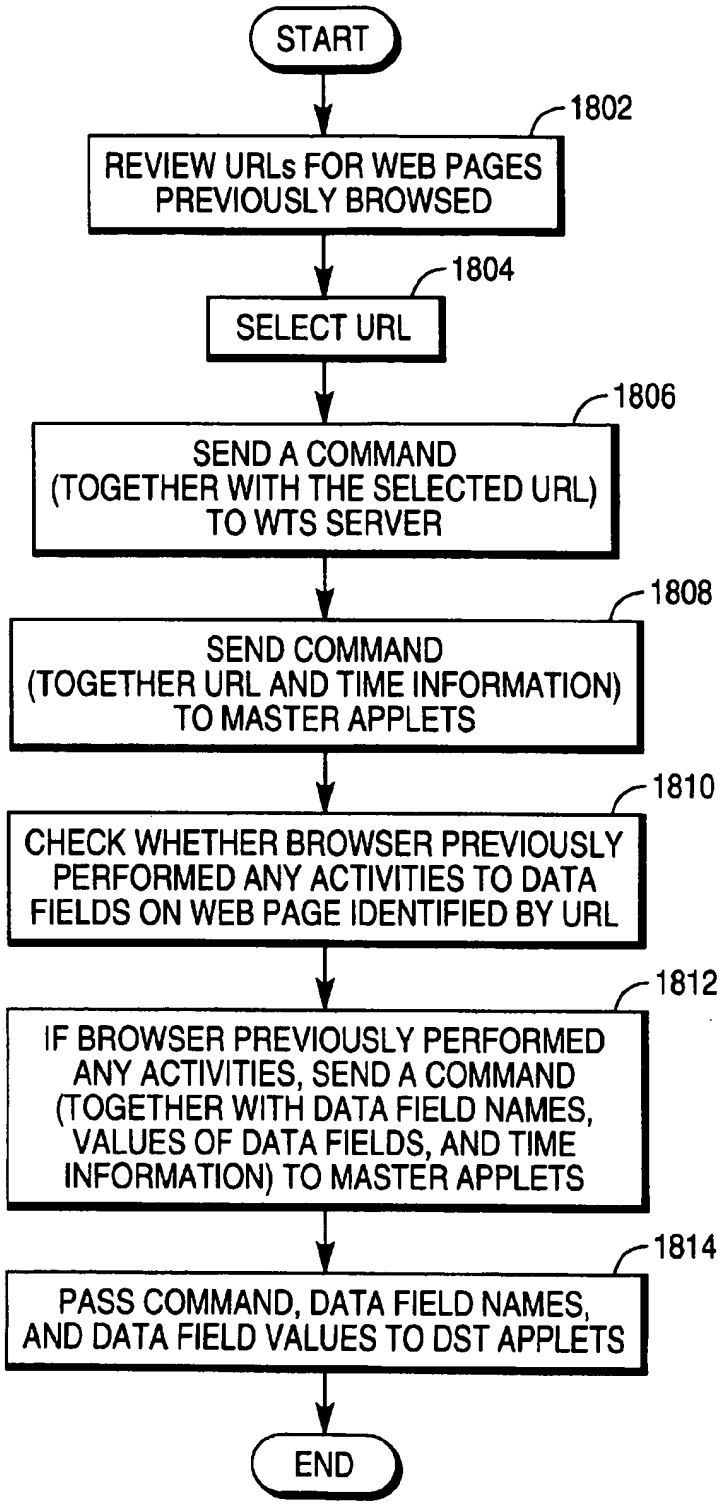
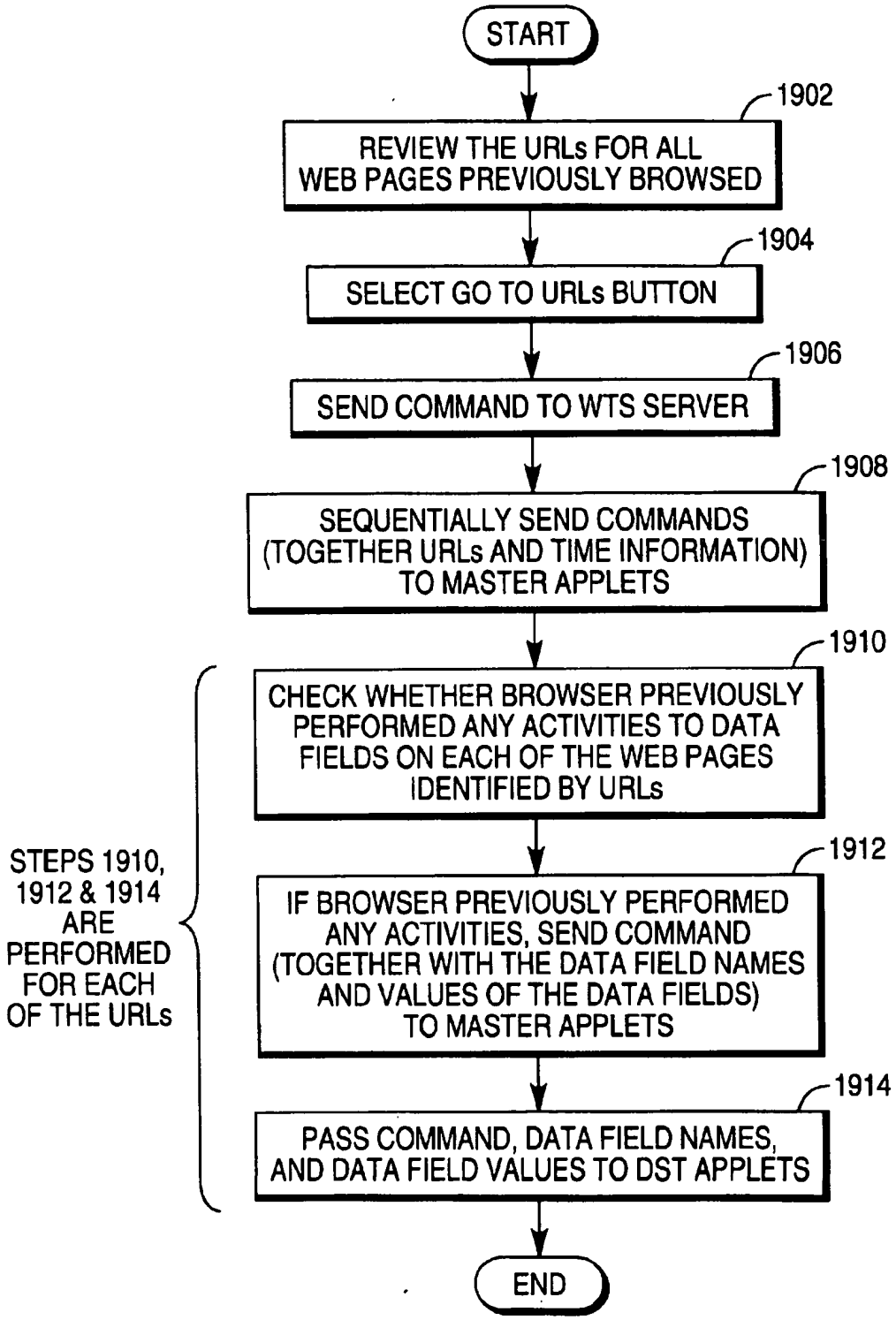

START
1802
REVIEW URLs FOR WEB PAGES PREVIOUSLY BROWSED
1804
SELECT URL
1806
SEND A COMMAND (TOGETHER WITH THE SELECTED URL) TO WTS SERVER
1808
SEND COMMAND (TOGETHER URL AND TIME INFORMATION) TO MASTER APPLETS
1810
CHECK WHETHER BROWSER PREVIOUSLY PERFORMED ANY ACTIVITIES TO DATA FIELDS ON WEB PAGE IDENTIFIED BY URL
1812
IF BROWSER PREVIOUSLY PERFORMED ANY ACTIVITIES, SEND A COMMAND (TOGETHER WITH DATA FIELD NAMES, VALUES OF DATA FIELDS, AND TIME INFORMATION) TO MASTER APPLETS
1814
PASS COMMAND, DATA FIELD NAMES, AND DATA FIELD VALUES TO DST APPLETS
END

FIG. 19


START
1902
REVIEW THE URLs FOR ALL WEB PAGES PREVIOUSLY BROWSED
1904
SELECT GO TO URLs BUTTON
1906
SEND COMMAND TO WTS SERVER
1908
SEQUENTIALLY SEND COMMANDS (TOGETHER URLs AND TIME INFORMATION) TO MASTER APPLETS
1910
CHECK WHETHER BROWSER PREVIOUSLY PERFORMED ANY ACTIVITIES TO DATA FIELDS ON EACH OF THE WEB PAGES IDENTIFIED BY URLs
1912
IF BROWSER PREVIOUSLY PERFORMED ANY ACTIVITIES, SEND COMMAND (TOGETHER WITH THE DATA FIELD NAMES AND VALUES OF THE DATA FIELDS) TO MASTER APPLETS
1914
PASS COMMAND, DATA FIELD NAMES, AND DATA FIELD VALUES TO DST APPLETS
END
STEPS 1910, 1912 & 1914 ARE PERFORMED FOR EACH OF THE URLs

METHOD FOR MONITORING USER INTERACTIONS WITH WEB PAGES FROM WEB SERVER USING DATA AND COMMAND LISTS FOR MAINTAINING INFORMATION VISITED AND ISSUED BY PARTICIPANTS

RELATED APPLICATIONS

This application relates to five other applications: (1) U.S. Ser. No. 08/944,757 filed Oct. 6, 1997, entitled "DEPENDABLE DATA ELEMENT SYNCHRONIZATION MECHANISM"; (2) U.S. Ser. No. 08/944,951 filed Oct. 6, 1997, entitled "DEPENDABLE DATA ELEMENT TRACKING MECHANISM"; (3) U.S. Ser. No. 08/944,121 filed Oct. 6, 1997, entitled "DEPENDABLE WEB PAGE SYNCHRONIZATION MECHANISM"; (4) U.S. Ser. No. 08/944,125 filed Oct. 6, 1997, entitled "MECHANISM FOR DEPENDABLY MANAGING WEB SYNCHRONIZATION AND TRACKING OPERATIONS AMONG MULTIPLE BROWSERS"; and (5) U.S. Ser. No. 08/944,124 filed Oct. 6, 1997, entitled "MECHANISM FOR DEPENDABLY ORGANIZING AND MANAGING INFORMATION FOR WEB SYNCHRONIZATION AND TRACKING AMONG MULTIPLE BROWSERS".

BACKGROUND OF THE INVENTION

The present invention relates generally to a method and apparatus for coordinating access to Internet web sites by a group of web browsers that are being run at a group of user terminals.

It is known that users can retrieve information from web sites (network sites) via the Internet. The basic model for retrieving information from web sites is user initiated information searching. Specifically, a user interacts with (via a terminal) a web browser to send a request to a web site. In response to the request, the web server for the web site retrieves the information requested and sends the web browser the information arranged in so called web page (HTML) format. One of the unique features of this model is the feature of "hyper-text links" embedded in web pages that have been retrieved. This feature enables a user in searching for information to "navigate" from one web page to another. In order to provide services (or assistance) to users (or customers) via the Internet, it is desirable to provide a mechanism to track activities performed to the web pages among a group of browsers.

One method of tracking web pages navigated is to install a monitoring program at a web site. When a terminal sends requests to a web site, the monitoring program at the web side collects the URLs for the requested web pages and sends the URLs to a server. However, under this method, the monitoring program is not always able to monitor the requests from the terminal, because when the terminal retrieves web pages from its browser cache space or from a proxy server, the requests are fulfilled locally and are never sent to the web site. As a result, the URLs are not accurately tracked.

Another method of tracking web pages navigated is to install a monitoring program together with a browser at a terminal. The monitoring program constantly communicates with the web browser. When the browser sends requests out, the monitoring program collects the URLs for the web pages requested by the browser and sends the URLs to a server. However, this method requires designing and installing monitoring programs that are capable of communicating with the different browsers. At the current time, different

web browsers are manufactured by a variety of vendors, including: Netscape®, Microsoft®, Sun Microsystem, IBM, and others. For a programmer to design such a monitoring program, it requires him/her to know the details of a proprietary web browser, and it may require updating the monitoring program whenever a proprietary web browser is updated. In addition, a monitoring program designed for a web browser manufactured by one vendor is typically not portable to another web browser manufactured by another vendor because browser interface mechanisms are proprietary. Moreover, users may perceive it as intrusive to be required to install a specialized application capable of collecting and reporting the information about the web pages retrieved from all other web sites.

Therefore, there is a need for an improved method to provide more dependable web page tracking.

There is another need for an improved method to provide web page tracking without requiring knowledge of the details about the web navigation software.

There is yet another need for an improved method to design web page tracking software that is portable to different software environments.

The present invention meets these needs.

SUMMARY OF THE INVENTION

In one aspect, the invention provides a method for tracking interactions with pages that have been loaded from a web server to a terminal, and for storing information about the interactions to a page tracking server. The method comprises the steps of:

(a) loading a first page from the web server, the first page being associated with a page locator for indicating a location of the first page in the web server, and the first page containing location information for indicating a location of a program;

(b) loading the program from the web server based on the location information, and executing the program;

(c) the program monitoring interactions with the page; and

(d) the program sending information about the interactions to the page tracking server.

In another aspect, the invention provides a method for tracking interactions with pages that have been loaded from a web server to a terminal, and for storing information about the interactions to a page tracking server. The method comprises the steps of:

(a) loading a page from the web server, the page being associated with a page locator for indicating a location of the page in the web server, and the page containing a program;

(b) executing the program;

(c) the program monitoring interactions with the page; and

(d) the program sending information about the interactions to the page tracking server.

The present invention also provides corresponding system for the respective aspects mentioned above.

BRIEF DESCRIPTION OF THE DRAWINGS

The purpose and advantages of the present invention will be apparent to those skilled in the art from the following detailed description in conjunction with the appended drawing, in which:

FIG. 1 shows a system includes N terminals, a network, and a web site, in accordance with the present invention;

FIG. 2 shows a situation where each of the N terminals has downloaded its respective Master Applets, DTS Applets, and SessionID Applet, in accordance with the present invention;

FIG. 3 shows the process the (consumer) Master Applet, DTS Applets, and SessionID Applet being downloaded into a terminal, in accordance with the present invention;

FIG. 4 shows the process the (consumer) Master Applet, DTS Applets, and SessionID Applet being invoked, in response to loading a subsequent web page, to perform the operations in accordance with the present invention, when these Applets have been previously downloaded and cached in a terminal;

FIG. 5 shows the process of the (consumer) Master Applet, DTS Applets, and SessionID Applet being invoked, in response to loading a subsequent web page, to perform the operations in accordance with the present invention, when both these Applets and the web page have been previously downloaded and cached in a terminal;

FIG. 6 shows a session table in greater detail, in accordance with the present invention;

FIG. 7 shows how an agent (or supervisor) can create a session interface by downloading an agent page (or a supervisor page) from administration page repository **149**, in accordance with the present invention.

FIG. 8A shows an agent session interface, in accordance with the present invention;

FIG. 8B shows a browser supervisor session interface, in accordance with the current invention;

FIG. 8C shows a supervisor agent session interface, in accordance with the present invention;

FIG. 9 shows a flowchart illustrating the operation of joining a session by an agent, in accordance with the present invention;

FIG. 10 shows a screen display containing two browse instances, in accordance with the present invention;

FIG. 11 shows a flowchart illustrating the operation of web page synchronization, in accordance with the present invention;

FIG. 12A shows a web page containing five data fields, in accordance with the present invention. FIG. 12B shows a web page that is similar to that of FIG. 12A, except that the data in one of the five data fields is changed, in accordance with the present invention;

FIG. 13 shows a flowchart illustrating the operation of data synchronization, in accordance with the present invention;

FIG. 14 shows a flowchart illustrating the operation of web page tracking, in accordance with the present invention;

FIG. 15 shows a flowchart illustrating the operation of data tracking, in accordance with the present invention;

FIG. 16 shows a flowchart illustrating the operation of joining a session by a supervisor, in accordance with the present invention.

FIG. 17, there shows three browser instances for a supervisor, in accordance with the present invention;

FIG. 18 shows a flowchart illustrating the operation of re-browsing a web page previously viewed in a session, in accordance with the present invention; and

FIG. 19 shows a flowchart illustrating the operation of re-browsing all web pages previously viewed in a session, in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the preferred embodiment(s) will be readily apparent to those skilled in the art, and the principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the invention. Thus, the present invention is not intended to be limited to the embodiment(s) shown, but is to be accorded with the broadest scope consistent with the principles and features disclosed herein.

Referring to FIG. **1**, there is shown an exemplary web page synchronization system **100**, in accordance with the present invention.

As shown in FIG. 1, the system includes N terminals (**104A**, . . . , **104K**, . . . , and **104N**), a network **129** (the Internet, or a combination of the Internet and an Intranet), and a web site **134**. Each of the terminals has a telephone set (**102A**, . . . , **102K**, . . . , or **102N**) installed in its vicinity. Each of the terminals can be a PC computer, a workstation, a Java station, or even a web TV system.

Web site **134** includes a WTS (Web Tracking and Synching) gateway **142**, a WTS server **144** containing a session table **145** and a user class table **147**, a database processing application **148**, an HTTP (Hyper Text Transfer Protocol) server **152**, and a hard disk unit **154** for storing consumer page repository **146**, administration page repository **149**, and database **156**. All the components in web site **134** can be installed in one or more computer systems. Each of the computer systems includes a processing unit (which may include a plurality of processors), a memory device, and a disk unit (which may include a plurality of disk sets).

Each of the terminals (**104A**, . . . , **104K**, . . . , or **104N**) includes a processor unit (not shown) and a memory area (**115A**, . . . , **115K**, . . . , **115N**), and runs a Java enabled web browser (**114A**, . . . , **114K**, . . . , or **114N**). Each of the memory area (**115A**, . . . , **115K**, . . . , or **115N**) is maintained by its respective browser (**114A**, . . . , **114K**, . . . , or **114N**). Via network **129**, each of the browsers (**114A**, . . . , **114K**, . . . , or **114N**) is able to send requests to and receive web pages from HTTP server **152**, and to display the web pages received at its respective terminal. Each of the browsers (**114A**, . . . , **114K**, . . . , or **114N**) is able to run a Master Applet (**124A**, . . . , **124K**, . . . , or **124N**), a set of DTS (Data Tracking and Synching) Applets, a SessionID Applet, and an Agent Applet. As shown in FIG. 1, these Applets are stored in consumer page repository **146** and can be downloaded from consumer page repository **146** and stored in the memory areas of the terminals (**104A**, . . . , **104K**, . . . , **104N**).

Referring to FIG. **2**, there is shown the situation where each of the terminals (**104A**, . . . , **104K**, . . . , or **104N**) has downloaded its respective Master Applets (**124A**, . . . , **124K**, . . . , or **124N**), DTS Applets (**126A**, . . . , **126K**, . . . , or **128N**), and SessionID Applet (**128A**, . . . , **128K**, . . . , **128N**), in accordance with the present invention.

In FIG. **2**, each of the (consumer) Master Applet (**124A**, . . . , **124K**, . . . , or **124N**) is primarily responsible for: (1) in response to loading each web page at its respective browser, opening a dedicated socket, and establishing a socket connection to WTS gateway **142** via network **129** for its respective browser (**114A**, . . . , **114K**, . . . , **114N**), (2) communicating with WTS server **144** via the socket connection, from which WTS server **144** is able identify the origin (i.e. which browser, which web page, etc.) of the commands and information that are being delivered through, (3) monitoring the activities of its respective browser, (4) sending the information about its respective browser's

activities to WTS server **144**, (5) receiving and processing the information about other browsers' activities, (6) via the socket connection, providing a single communication path to WTS server **144** for DTS Applets (**126A**, . . . , **126K**, . . . , or **126N**), SessionID Applets (**128A**, . . . , **128K**, . . . , or **128N**), or any other consumer Applets embedded on the same page with the Master Applet, (7) sending commands to WTS server **144** to request services, for itself and for DTS Applets (**126A**, . . . , **126K**, . . . , or **126N**), SessionID Applets (**128A**, . . . , **128K**, . . . , or **128N**) , or any other consumer Applets embedded on the same page with the Master Applet, and (8) sending user class information together with the commands, to indicate that its respective browser is a consumer user.

Each set of DTS Applets (**126A**, . . . , **126K**, . . . , or **126N**) contains one or more individual DTS Applets, which are primarily responsible for: (1) displaying and monitoring the data activities (data inputs or data updates of data fields) on web pages that are being displayed by its respective browser, (2) sending the data activities to WTS server **144** via its respective Master Applet, (3) receiving the data activities from other browsers via its respective Master Applet, and (4) processing the data activities from other browsers for the web pages that are being displayed by its respective browser.

Each of the SessionID Applets (**128A**, . . . , **128K**, . . . , or **128N**) is responsible for retrieving, and for displaying on a web page the current SessionID.

As shown in administration page repository **149**, Agent Applet (or Supervisor Applet) is responsible for creating a session interface, joining, monitoring, and controlling a session through the session interface. The(administration) Master Applet is primarily responsible for: (1) opening a dedicated socket, and establishing a socket connection to WTS gateway **142** via network **129** for the session interface created by Agent Applet, Supervisor Applet, or any other administration Applets embedded on the same web page with the Master Applet, (2) communicating with WTS server **144** via the socket connection, from which WTS server **144** is able identify the origin (i.e. from which session interface) of the commands and information that are being delivered through, (3) via the socket connection, providing a single communication path to WTS server **144** for Agent Applet, Supervisor Applet, or any other administration Applets embedded on the same web page with the Master Applet, and (4) sending user class information together with the commands, to indicate that its respective browser is an administration user.

WTS gateway **142** is responsible for maintaining all socket connections between Master Applets and WTS server **144**. The connections between Master Applets and WTS gateway **142** take place using standard sockets. The connection between WTS gateway **142** and WTS server **144** takes place using RMI (Remote Method Invocation).

WTS server **144** is responsible for: (1) managing and tracking the activities of all browsers participating in active sessions, exemplary activities including: loading of, interacting with, and unloading of web pages, (2) recording the information about the activities, (3) managing the synchronization of the activities for all browsers participating in the active sessions, (4) creating a session when a consumer user (via a browser) sends a request to web site **134** for the first time, (5) defining the session length intervals, (6) purging sessions that have been inactive for more than the specified session length intervals, (7) adding and deleting participants to a session, and (8) providing services to all commands from consumer Applets, such as: (consumer) Master Applet, DTS Applets, SessionID Applets, and administration Applets, such as: (administration) Master Applet, Agent Applets, and Supervisor Applet.

Consumer page repository **146** stores the web pages and Applets for consumers. Consumer Applets can be selectively embedded into consumer web pages. Exemplary consumer Applets include: (consumer) Master Applet, DTS Applets, SessionID Applet, etc.

Administration page repository **149** stores the web pages and Applets for call center administration users, including: administrator, supervisor, agent, etc. Administration Applets can be selectively embedded into administration web pages. Exemplary administration Applets include: (administration) Master Applet, Agent Applet, Supervisor Applet, etc.

To better describe the present invention, the Applets stored in (or downloaded from) repository **146** can be referred to as consumer Applets, and the Applets stored in (or downloaded from) repository **149** can be referred to as administration Applets. For example, the Master Applet stored in (or downloaded from) repository **146** can be referred to as consumer Master Applet, and the Master Applet stored in (or downloaded from) repository **149** can be referred to as administration Master Applets. HTTP server **152** contains a security application that allows consumer users to get access only to the web pages stored in consumer page repository **146**, and allows administration users (such as administrator, supervisor, agent, etc.) to get access to the web pages stored in both consumer page repository **146** and administration page repository **149**.

Session table **145** is responsible for maintaining the information for all active sessions.

Class table **147** is responsible for keeping records of user classes assigned to different users. Listed are exemplary user classes: administrator, supervisor, agent, and consumer.

Based on user classes (administrator, supervisor, agent, and consumer), WTS server **144** provides the following services:

(1) creating a session (consumer);

(2) storing data received from a session participant (supervisor, agent, and consumer);

(3) listing active sessions (administrator and supervisor);

(4) listing the information associated with active sessions (administrator, and supervisor);

(5) listing current users (administrator);

(6) joining a session (supervisor and agent);

(7) terminating a session (supervisor);

(8) monitoring a session (supervisor and agent);

(9) configuring a session parameters (administrator); and

(10) sending commands and information to a consumer Master Applet or an administration Master Applet in a participating browser (supervisor, agent, and consumer).

Database **156** is responsible for storing data collected in session table **145**.

HTTP server **152** is responsible for processing the requests issued by one of the web browsers, retrieving the web pages from either consumer page repository **146** or administration page repository **149**, and sending the web pages to the browsers that have generated these requests.

Database processing application **148** is responsible for writing the data collected in session table **145** into database **156**.

Referring to FIG. **3**, there is shown the process of how the (consumer) Master Applet, DTS Applets, and SessionID

Applet being downloaded into terminal **104A** from HTTP server **152** in response to loading an initial web page, and then being invoked to perform the operations in accordance with the present invention.

As shown in FIG. **3**, a (consumer) Master Applet, a set of DTS Applets, and a SessionID Applet are embedded into web page **204** by using a set of applet tags **208**. Web page **204** is associated with a specific URL indicating the location of web page **204** in HTTP server **152**.

As indicated by dotted line (**1**), web browser **114A** sends a request including the URL of web page **204** to HTTP server **152** via network **129**. As indicated by dotted line (**2**), in response to the request, HTTP server **152** retrieves web page **204** from consumer page repository **146** and sends it to web browser **114A** via network **129**. Web page **204** contains a set of applet tags **208**, which indicate the location of Master Applet, DTS Applets, and SessionID Applet in HTTP server **152**. As indicated by dotted line (**3**), web browser **114A** loads web page **204**. As indicated by dotted line (**4**), since Master Applet, DTS Applets, and SessionID Applet have not been downloaded, web browser **114A** sends requests via network **129**, to download these Applets based on applet tags **208**. As indicated by dotted line (**5**), HTTP server **152** sends Master Applet, DTS Applets, and SessionID Applet to browser **114A** via network **129**. As indicated by dotted line (**6**), browser **114A** stores Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** into memory area **115A** of terminal **104A**, and initializes and invokes these Applets. After being invoked, these Applets are running together with web browser **114A**, to monitor and process the activities for which they are assigned to be responsible. As indicated by line (**7**), Master Applet **124A** opens a dedicated socket and establishes a socket connection to WTS gateway **142** for browser **114A** and web page **204**. Via the socket connection, Master Applet **126** sends WTS server **144** a command, together with an ID unique to browser **114A**. In response to the command from Master Applet **126**, WTS server **144** creates a session for browser **114A** based on the unique ID, and issues a time stamp (loading time) indicating the time at which the command was received, and stores the URL and time stamp of web page **204** into the session created for browser **114**. As will see in the description in connection with FIG. **6** following, the URL, command, and loading time are stored in a URL history list and a command list created for the session.

Referring to FIG. **4**, there is shown the process of the (consumer) Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** being invoked, in response to loading a subsequent web page **214** (subsequent to web page **204**), to perform the operations in accordance with the present invention, when these Applets have been previously downloaded and cached in terminal **104A**.

As indicated by dotted line (**1**), to download web page **214**, web browser **114A** sends a request including the URL of web page **214** to HTTP server **152** via network **129**. Before loading web page **214**, the following events occur: (a) browser **114A** instructs Master Applet **124A** to run a stop routine, (b) via the socket connection established for browser **114A** and web page **204**, Master Applet **124** sends a command to inform WTS server **144** that web page **204** has been unloaded, and disconnects the socket connection established for browser **114A** and web page **204**, (c) WTS server **144** issues a time stamp (unloading time) indicating the time the command was received, and (d) records the URL and the time stamp of web page **204** into the session created for browser **114A**. As will be seen in the description in connection with FIG. **6**, following, URL, command, and unloading time are stored in a URL history list and a command list created for the session. As indicated by dotted line (**2**), HTTP server **152** retrieves web page **214** from consumer page repository **146** and sends it to browser **114A**. Like web page **204**, web page **214** contains a set of applet tags **208** for indicating the location of Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A**. As indicated by dotted line (**3**), web browser **114A** loads web page **214**. As indicated by dotted line (**4**), in response to the loading of web page **214**, web browser **114A** locates Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** (based on the indication of applet tags **208**) that are cached by browser **114A** in memory area **115A**, and initializes these Applets and then invokes them. As indicated by line (**5**), Master Applet **124A** opens a dedicated socket and establishes a socket connection to WTS gateway **142** for browser **114A** and web page **214**. Via the socket connection established for browser **114A** and web page **214**, Master Applet **126A** sends a command, together with the ID unique to browser **114A** and the URL of web page **214**, to inform WTS server **144** that web page **214** has been loaded. WTS server **144** issues a time stamp (loading time) indicating the time the command was received and stores the URL and time stamp of web page into the session created for browser **114A**. As will be seen in the description in connection with FIG. **6**, following, URL, command, and loading time are stored in a URL history list and a command list created for the session.

Referring to FIG. **5**, there is shown the process of the (consumer) Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** being invoked, in response to loading a subsequent web page **224** (subsequent to web page **214**), to perform the operations in accordance with the present invention, when both these Applets and web page **224** have been previously downloaded and cached by browser **114A** in terminal **104A**.

As indicated by dotted line (**1**), web browser **114A** loads web page **224** cached in memory area **115A** maintained by browser **114A**. Like web pages **204** and **214**, web page **224** contains a set of applet tags **208** indicating the location of Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A**. Before loading web page **224**, the following events occur: (a) browser **114A** instructs Master Applet **124A** to run a stop routine, (b) via the socket connection established for browser **114A** and web page **214A**, Master Applet **124A** sends a command to inform WTS server **144** that web page **214** has been unloaded, and disconnects the socket connection established for browser **114A** and web page **214**, (c) WTS server **144** issues a time stamp (unloading time) indicating the time the command was received, and (d) WTS server **144** records the URL and time stamp of web page **214** into the session created for browser **114A**. As will be seen in the description in connection with FIG. **6**, following, the URL, command, and unloading time are stored in a URL history list and a command list created for the session. As indicated by dotted line (**2**), in response to the loading of web page **224**, browser **114A** locates Master Applet **124A**, DTS Applets **126A**, SessionID Applet **128A** that have been cached by browser **114A** in memory area **115A** in terminal **104A**, and initializes and invokes these Applets. As indicated by line (**3**), Master Applet **124A** opens a dedicated socket and establishes a socket connection to WTS gateway **142** for browser **114A** and web page **224**. Via the socket connection established for browser **114A** and web page **224**, Master Applet **126A** sends a command, together with the ID unique to browser **114A** and the URL of web page **224**, to inform WTS server **144** that web page **224** has been loaded. WTS server **144** issues a time stamp

(loading time) indicating the time the command was received and stores the URL and time stamp into the session created for browser **114**. As will be seen in the description in connection with FIG. **6**, following, the URL, command, and loading time are stored in a URL history list and a command list created for the session.

In the example shown in FIG. **5**, it should be appreciated that even through no request arrives at HTTP server **144** when web page **224** is loaded from cached memory in terminal **104**A, Master Applet **124**A still sends browsing activities to WTS server **144**.

It should be noted that the processes shown in FIGS. **3–5** of loading and invoking Master Applet, DTS Applets, and SessionID Applet for terminal **104**A can also be used for terminals **104**K, . . . , **104**N.

In FIGS. **3–5**, Master Applet, DTS Applets, and SessionID Applet are all embedded into web pages **204**, **214**, and **224**. However, it should be noted that not all the Applets are required to be embedded into a web page. Depending on the desired functions to be performed, respective Applets can be selectively embedded into a web page by selectively setting applet tags in the web page. For example, if data synchronization and tracking of individual elements are not needed, the applet tags for linking DTS Applets can be eliminated from the web page. By the same token, if additional functions are needed, additional applet tags can be added into the web page to link additional Applets.

Referring to FIG. **6**, there is shown session table **145** (see FIG. **1**) in greater detail, in accordance with the present invention.

While browsers at their respective terminals are browsing through the web pages in web site **134**, WTS server **144** collects and analyzes the information about the interactions between all browsers and the web pages that have been downloaded to the browsers from web site **134**. One difficulty in collecting and analyzing such information is that browsing individual web pages in web site **134** is a stateless process. More specifically, web site **134** receives a sequence of requests from different browsers, and sends the respective web pages to the respective browsers in response to the sequence requests. Since in processing the requests from an individual browser, web site **134** does maintain a constant connection to the same browser to keep an one-to-one relationship, web site **134** has no control over, or maintain data on, the sequences of the requests from the browsers.

To meaningfully collect and analyze the information about the interactions between the browsers and web pages, a session is defined as a collection of web page interactions that occur over a given period of time from a specific browser. A session is created when a browser first hits web site **134**, and a session window (or session length interval) is defined for the session. If activities from a specific browser (identified by an ID unique to the browser, issued by a respective Master Applet) does not occur within the session window, the session is terminated and cleaned up by WTS server **144**. A session window is refreshed (reset to time zero) each time the information about the associated browser is sent to WTS server **144**. For example, if a session window is defined as 15 minutes, as long as the associated terminal has some activity every 15 minutes, the session will remain open. After 15 minutes of inactivity, the session is terminated and purged. A subsequent request from the same terminal will cause a new session to be created. After a session has been created for a terminal, one or more other terminals can join the session.

As shown in FIG. **6**, session table **145** includes M Session IDs created for M sessions respectively. Each of the session ID is associated with: (1) a session list for maintaining information about a session, (2) a participant list for maintaining information about all participant browsers in a session (note: when a session is first created, it only contains one participant), (3) a URL history list for maintaining information about all web pages visited by all participants in a session, (4) a data list for maintaining information about the data fields on the web pages visited by all participants in a session, and (5) a command list for maintaining information about all commands issued to WTS server **144** by the various participants in a session.

Typical items in a session list are: (1) SessionID for identifying a session, (2) UserName for indicating the actual name for whom the session is created, (3) StartTime for indicating the time of starting the session, (4) StopTime for indicating the time of stopping the session, and (5) SessionNotes for recording the notes of the session.

Typical fields contained in a participant list are: (1) SessionID for linking the participant list to a session, (2) ParticipantID for identifying a participant, (3) ParticipantAddresses for indicating a participant's IP address, (4) Class for indicating the user class of the participant (customer, agent, supervisor, administrator, etc.) and (5) Direction for indicating the synchronization direction for the participant browser.

Typical fields contained in a URL history list are: (1) SessionID for linking the URL history list to a session, (2) PageURL for indicating the URL of a web page visited, (3) ParticipantID for identifying a participant who visited the web page, (4) LoadingTime for indicating the loading time of the web page, and (4) UnloadingTime for indicating the unloading time of the web page.

Typical fields contained in a data list are: (1) SessionID for linking the data list to a session, (2) WasRelayed for indicating if this data field has been broadcasted, (2) FieldName for indicating the actual name of the data field, (3) DataName for indicating the name of the data field displayed on a web page, (4) DataValue for indicating the value of the data field, (5) TimeStamp for indicating the time at which this data field is updated, (6) URL for indicating the web page on which the data field was displayed, and (7) ParticipantID for indicating the participant browser who updated this data field.

Typical fields contained in a command list are: (1) SessionID for linking the data list to a session, (2) Command for indicating the specific command executed (loading a page, unloading a page, changing a data field, etc.), (3) URL for indicating the web page to which the command operated, (4) FieldPoint for indicating the data field to which the command operated, and (5) TimeStamp for indicating the time at which command was executed.

Before a session is purged from session table **145**, database processing application **147** stores the associated session list, URL history list, and command list to database **156**. The data contained in these three lists can be late used by data warehouse integration applications.

Referring to FIG. **7**, there is shown an operation for creating a session interface for an agent (or a supervisor) by downloading an agent page (or a supervisor page) from administration page repository **149**, in accordance with the present invention. In the example shown in FIG. **7**, it is assumed that administration user class (either agent user class or supervisor user class) is assigned to terminal **104**N, so that the security application in HTTP server **152** grants the access to the web page stored in both consumer page repository **146** and administration page repository **149**.

At step **702**, an agent at terminal **144**N types in an agent URL at terminal **104**N, and browser **114**N sends the URL to

HTTP server **152**, to retrieve an agent page, in which an (administration) Master Applet and an Agent Applet are embedded. For a supervisor, he/she types in a supervisor URL at terminal **104N**, and browser **114N** sends the URL to HTTP server **152**, to retrieve a supervisor page, in which an (administration) Master Applet and a Supervisor Applet are embedded.

At step **704**, HTTP server **152** retrieves the agent page (or a supervisor page) from administration page repository **149** and sends it to browser **114N**.

At step **706**, browser **114N** downloads the agent page, in which a Master Applet (administration Master Applet) and an Agent Applet are embedded; or downloads the supervisor page, in which a Master Applet (administration Master Applet) and a Supervisor Applet are embedded.

At step **708**, browser **114N** downloads the Master Applet and Agent Applet from HTTP server **152**, initializes and invokes these Applets; or downloads the Master Applet and Supervisor Applet from HTTP server **152**, initializes and invokes these Applets.

At step **710**, Master Applet opens a dedicated socket, establishes a socket connection to WTS gateway **142**, and sends an ID unique to browser **114N** to WTS server **144**. WTS server **144** is able to identify browser **114N** based on the unique ID.

At step **712**, Agent Applet creates an agent session interface **800A** shown in FIG. **8A** for the agent user; or Supervisor Applet creates a supervisor session interface **800B** shown in FIG. **8B** for the supervisor agent.

Referring to FIG. **8A**, there is shown an agent session interface **800A** created for an agent at step **712**, in accordance with the present invention.

As shown in FIG. **8A**, the session interface contains a text box **804** for entering a session ID, a Join session button **806** for joining a session identified by the session ID, a drop button **808** for leaving a session, a leader check box **810** (selecting of which designates a browser as a leading browser in synchronization), a follower check box **812** (selecting of which designates a browser as a following browser in synchronization), a scrollable list box **816** for displaying the information contained in the participant list associated with a selected session, a scrollable list box **818** for displaying the information in an identified URL history list, and a text box **820** for displaying the information in an identified data list. If both the leader and follower check boxes **810** and **812** are selected in the agent session interface, browser **114A** acts as both leading and following browser in synchronization.

Referring to FIG. **8B**, there is shown a browser supervisor session interface **800B** created for a supervisor at step **712**, in accordance with the current invention.

As shown in FIG. **8B**, the session interface contains a scrollable list box **832** for displaying session IDs of all active sessions in session table **145** and for selecting one of the session IDs, a text box **834** for displaying relevant statistics of WTS server **144**, a multi column scrollable list box **836** for displaying details about the session selected in scrollable list box **832**, a select session button **838** for selecting a session from scrollable list box **832**. By using the information in scrollable list box **832**, a supervisor agent can monitor all active sessions. By using the information in multi column scrollable list box **836**, a supervisor can monitor operational status of a session selected from scrollable list box **832**, including: (1) whether this session is being helped by an agent, (2) user name, and (3) agent ID. By selecting select session button **838**, a supervisor can create an agent session interface as shown in FIG. **8C**.

Referring to FIG. **8C**, there is shown a supervisor agent session interface **800C**, in accordance with the present invention.

Referring to FIG. **9**, there is shown a flowchart illustrating the operation of joining a session by an agent, in accordance with the present invention.

In the example shown in FIG. **9**, it is assumed that: (1) a consumer at terminal **104A** is browsing web pages from consumer page repository **146** via browser **114A**, (2) session list **1** shown in FIG. **6** has been created for browser **114A**, (3) an agent class has been assigned to browser **114N**, (4) agent session interface **800A** shown in FIG. **8A** has been displayed on terminal **104N**; (5) a (administration) Master Applet and Agent Applet have been previously downloaded into browser **114N**, (6) a dedicated socket connection has been established for session interface **800A** displayed at terminal **104N** by the (administration) Master Applet, and (7) the agent at terminal **104A** is on duty at a call center.

As shown in FIG. **9**, at step **902**, the consumer is browsing a web page at terminal **104A**. On the web page, SessionID Applet **128A** displays the current session ID. A call center telephone number the consumer can call is also displayed on the web page.

At step **904**, the consumer is connected to the call center by dialing the telephone number via telephone **102A** (see FIG. **1**), and the call is directed by the call center to the agent.

At step **906**, the consumer tells, via telephone **102A** (see FIG. **1**), the agent the current session ID displayed. It should be noted that, instead of using the telephone, the agent can be informed of the current session ID by alternative methods. For example, the consumer can enter his/her telephone number into a special web page that contains the caller ID of the consumer along with the current session ID. This information can be stored into a special lookup table that can be used by the agent to lookup the current session ID.

At step **908**, at terminal **104N**, the agent types the current session ID into text box **804** (see FIG. **8A**).

At step **910**, in response to a loss of focus or a pressing of the Enter key, through the socket connection established for agent session interface **800A** displayed on terminal **104N**, the (administration) Master Applet at terminal **104N** sends a command to WTS server **144**, to retrieve the information in participant list **1**, URL history list **1**, and data list **1** (see FIG. **6**) for the Agent Applet.

At step **912**, WTS server **144** sends the information requested to the Agent Applet (via the Master Applet).

At step **914**, the Agent Applet at terminal **104N** displays some information from participant list **1** and URL history list **1** in (participant) scrollable list box **816** and (URL history) scrollable list box **818**, respectively.

At step **916**, the agent selects join button **806** in agent session interface **800A** displayed on terminal **104N**.

At step **918**, in response to the selection at step **916**, through the socket connection which has been established for agent session interface displayed on terminal **104N**, the (administration) Master Applet sends WTS server **144** a command to join the selected session. Based on the identification associated with the socket connection, WTS server is able to generate a ParticipantID for browser **114N** and to find the ParticipantAddress for terminal **104N**.

At step **920**, WTS server **144** stores the ParticipantID and ParticipantAddress into participant list **1**. At this step, participant list **1** includes two participant records (two rows) containing the PaticipantIDs for browsers **114A** and **114N** respectively.

At step **922**, at terminal **104N**, the agent selects: leading check box **810** or following check box **812**, or both of them.

By only selecting leader check box **810**, the activities at terminal **104N** are synchronized at terminal **104A**, but not other way around. By only selecting follower check box **812**, the activities at terminal **104A** are synchronized at terminal **104N**, but not other way around. By selecting both leader and follower check boxes **810** and **812**, the activities at terminals **104A** and **104N** are synchronized with each other (bi-directional synchronization). In response to the selection(s), through the socket connection which has been established for agent session interface **800A**, the (administration) Master Applet sends WTS server **144** a command designating the synchronization direction. WTS server **144** stores the synchronization direction information into the Direction fields of the two records in participation list **1**. In this example, it is assumed that the bi-directional synchronization has been selected for terminals **104A** and **104N**.

At step **924**, WTS server **144** sends the (administration) Master Applet the URL of the web page being currently browsed at terminal **104A**.

At step **926**, the Agent Applet at terminal **104N** opens a browser window **1004** (a second browser instance) as shown in FIG. **10**.

At step **928**, browser **114N** downloads the web page identified by the URL from consumer page repository **146**, and displays it in browser window **1004**. A (consumer) Master Applet, a set of DTS Applets, and a SessionID Applet are embedded in the web page downloaded.

At step **930**, browser **114N** downloads (consumer) Master Applet **124N**, set of DTS Applets **126N**, and SessionID Applet **128N**.

At step **932**, the web pages displayed in second browser window **1004** at terminal **104N** are being synchronized with the web pages being displayed at terminal **104A**.

After step **932**, if the agent (the first agent) at terminal **104A** needs assistance from another agent (the second agent) at terminal **104K**, the first agent can call the second agent and tell him/her the current session ID. The second agent can then join the current session using an agent session interface as shown in FIG. **8A** displayed at terminal **104K**.

Referring to FIG. **10**, there is shown a screen display containing two browser instances (**800A** and **1004**) at terminal **104N**, in accordance with the present invention.

As shown in FIG. **10**, at terminal **104N**, the first browser instance provides an agent session interface **800A** to control and monitor the current session, and the (administration) Master Applet for agent session interface **800A** establishes and maintains a socket connection for agent session interface **800A**. The second browser instance provides a browser window **1004** to display the web pages being synchronized. (Consumer) Master Applet **124N** establishes and maintains a socket connection for each web page displayed in browser window **1004**.

Referring to FIG. **11**, there is shown a flowchart illustrating the operation of web page synchronization, in accordance with the present invention.

In the example shown in FIG. **11**, it is assumed that: (1) a consumer at terminal **104A** is browsing web pages from consumer page repository **146** via browser **114A**, (2) a session has been created for browser **114A**, (3) session list **1** and participant list **1** shown in FIG. **6** have been created for the session, (4) bi-directional synchronization has been selected for terminal **104A** and all participant terminals, and (5) the (consumer) Master Applet, DTS Applets, and SessionID Applet have been downloaded into browser **104A** and all participant browsers.

As shown in FIG. **11**, at step **1104**, browser **114A** loads a web page either from consumer page repository **146** or from memory area **115A** in terminal **104A**. If Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** had not been download to browser **114A**, browser **114A** would download these Applets from consumer page repository **146**. However, in this example, these Applets are assumed to be downloaded.

At step **1106**, in response to the loading of the web page, browser **114A** initializes and invokes Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A**.

At step **1108**, Master Applet **124A**: (1) opens a dedicated socket, and establishes a socket connection to WTS gateway **142** for browser **114A** and the web page loaded, and (2) via the socket connection, sends WTS server **144** a command together with an ID unique to browser **114A** and the URL of the web page loaded. Based on the unique ID, WTS server is able to identify the session created for browser **114A**.

At step **1110**, WTS server **144** identifies the session for browser **114A**.

At step **1112**, WTS server **144** locates all IP addresses assigned to participant terminals in participant list **1** (shown in FIG. **6**), and sends a command, together with the URL, to all the participant terminals (except that WTS server **144** does not sent the URL to terminal **104A**, because the URL is originated from terminal **104A**).

At step **1114**, upon receiving the command, the (consumer) Master Applets in the participant terminals initialize themselves and pass the URL to their respective browsers.

At step **1116**, the respective browsers in the participant terminals download and display the web page according to the URL.

It should be noted that, like terminal **104A**, each of the participant terminals (at which agent session interface is displayed) can lead the page synchronization using the operation shown in FIG. **11**.

Referring to FIG. **12A**, there is shown a web page containing five data fields, specifically: name **1202**, time period **1204**, account balance **1206**, payment **1208**, comments **1210**, a text box **1212** for displaying the current session ID, and a text box for displaying the call center number the consumer can call, in accordance with the present invention.

Referring to FIG. **12B**, there is shown a web page that is similar to that of FIG. **12A**, except that the data in the field of name **202** is changed from Susan King to Sue Grant and the changes are synchronized at a participant terminal, in accordance with the present invention.

Referring to FIG. **13**, there is shown a flowchart illustrating the operation of data synchronization, in accordance with the present invention.

In the example shown in FIG. **13**, it is assumed that: (1) a customer at terminal **104A** is browsing web pages via browser **114A**, (2) a session has been created for terminal **104A**, (3) session list **1** and participant list **1** shown in FIG. **6** has been created for the session, (4) terminal **104N** is one of the participants, (5) web page **1200** containing five data fields shown in FIG. **12A** is displayed on terminals **104A** and all participant terminals, (6) a bi-directional synchronization has been selected for terminal **104A** and all participant terminals, (7) the (consumer) Master Applet, DTS Applets, and SessionID Applet have been downloaded to browser **114A** and the browsers at all participant terminals, (8) the DTS Applets contains five individual Applets: DTS $Applet_1$, DTS $Applet_2$, DTS $Applet_3$, DTS $Applets_4$, and DTS $Applet_5$, (9) these five individual DTS Applets are respectively responsible for monitoring and processing the events occurred on the five data fields of web page **1200** shown in

FIG. 12A, (10) (consumer) Master Applet 124A has established a dedicated socket connection to WTS gateway 142 for web page 12A displayed at terminal 104A, and (11) the customer at terminal 104A wants to make changes to name field 1202 from Susan King to Sue Grant.

As shown in FIG. 13, at step 1304, the customer changes the name in name field 1202 from Susan King to Sue Grant.

At step 1306, in response to a loss of focus on name field 1202 or pressing the Enter key, DTS Applet$_1$ detects the change and passes the change to Master Applet 124A.

At step 1308, via the dedicated socket connection, Master Applet 124A sends WTS server 144 a command together with the change of name field 1202. Since this change is passed to WTS server 144 via the dedicated socket connection established for web page 1200, WTS server 144 is able to recognize the origin of the command, web page 1200, and the name field upon which the change was made.

At step 1310, WTS server 144 identifies the session created for browser 114A.

At step 1312, WTS server 144 locates the IP addresses assigned to participant browsers in participant list 1 and sends a command (together with the change of name field 1202) to the Master Applets in all participant terminals (except that WTS server 144 does not send the command and change to browser 114A, since this change originated from browser 114A).

At step 1314, upon receiving the command, the (consumer) Master Applets (including Master Applets 124N) pass the change of name field 1200 to their respective DTS Applets, including the DTS Applet$_1$ at browser 114N.

At step 1316, the DTS Applet$_1$, display the update "Susan Grant" into the name fields on respective web page 1200 displayed on the respective terminals, including terminal 104N.

It should be noted that the operation shown in FIG. 13 can be used to perform data synchronization for the other four data fields on web page 1200 shown in FIG. 12A.

It should also be noted that the data field synchronization can also be performed at terminal 104N. For example, as shown in FIG. 12B, when the agent at terminal 104N enters comments of "Account's name had been changed" to comments field 1210' on web page 1200', this updates will be displayed in comments field 1210 at terminal 104A, by using the operation shown in FIG. 13.

Referring to FIG. 14, there is shown a flowchart illustrating the operation of web page tracking, in accordance with the present invention.

In the example shown in FIG. 14, it is assumed that: (1) a customer at terminal 104A is browsing web pages via browser 114A, (2) a session has been created for terminal 104A and all participant terminals, (3) session list 1, participant list 1, and URL history list 1 shown in FIG. 6 have been created for the session, (4) bi-directional synchronization has been selected for terminal 104A and all participant terminals, and (5) the (consumer) Master Applet, DTS Applets, and SessionID Applet have been downloaded into terminals 104A and all participant terminals.

As shown in FIG. 14, at step 1404, browser 114A downloads a web page from either the consumer page repository 146 or memory area 115A of terminal 104A. If Master Applet 124A, DTS Applets 126A, and SessionID Applet 128A had not been download to terminal 104A, browser 114A would download them from HTTP server 152. However, in this example, these Applets have been downloaded.

At step 1406, web browser 114A initializes and invokes Master Applet 124A, DTS Applets 126A, and SessionID Applet 128A.

At step 1408, Master Applet 124A opens a dedicated socket and establishes a socket connection to WTS gateway 142 for web browser 1 14A and the web page loaded. Master Applet 124A then sends WTS server 144 a command, together with: (1) an ID unique to browser 114A, and (2) the URL of the web page loaded. When commands and URL are delivered through this socket connection, WTS server 144 is able to recognize the origin of the commands and URL.

At step 1410, WTS server 144 identifies the session ID for browser 114A.

At step 1412, WTS server 144 locates the session list 1 and URL history list 1.

At step 1414, WTS server 144 issues a time stamp (loading time) for indicating the time at which the command was received, and stores the URL and time stamp to URL history list 1.

At step 1416, browser 114A sends WTS server 144 a request to load a subsequent web page.

At step 1418, before loading the subsequent web page, via the socket connection, Master Applet 124A sends WTS server 144 a command, together with the URL, to inform WTS server 144 that the current web page has been unloaded.

At step 1420, WTS server 144 identifies the session for terminal 104A.

At step 1422, WTS server 144 locates the session list 1 and URL history list 1.

At step 1424, WTS server 144 issues a time stamp (unloading time) for indicating the time at which the command was received, and stores the URL and time stamp to URL history list 1.

At step 1426, Master Applet 124A disconnects the socket connection for the web page that has been unloaded.

Referring to FIG. 15, there is shown a flowchart illustrating the operation of data tracking, in accordance with the present invention.

In the example shown in FIG. 15, it is assumed that: (1) a customer at terminal 104A is browsing web pages via browser 114A, (2) a session has been created for terminal 104A, (3) session list 1 and participant list 1 shown in FIG. 6 has been created for the session, (4) terminal 104N is one of the participants, (5) web page 1200 containing five data fields shown in FIG. 12A is displayed on terminals 104A and all participant terminals, (6) a bi-directional synchronization has been selected for terminal 104A and all participant terminals, (7) the (consumer) Master Applet, DTS Applets, and SessionID Applet are downloaded to terminal 104A and all participant terminals, (8) the DTS Applets contains five individual Applets: DTS Applet$_1$, DTS Applet$_2$, DTS Applet$_3$, DTS Applets$_4$, and DTS Applet$_5$, (9) these five individual DTS Applets are respectively responsible for displaying, monitoring and processing the events occurred on the five data fields of web page 1200 shown in FIG. 12A, (10) Master Applet 124A has established a dedicated socket connection to WTS server 144 for web page 1200 displayed on terminal 104A, and (11) the customer at terminal 104A wants to make changes to name field 1202 from Susan King to Sue Grant.

As shown in FIG. 15, at step 1504, the customer changes the name in name field 1502 from Susan King to Sue Grant.

At step 1506, in response to a loss of focus on name field 1202 or pressing the Enter key, DTS Applet$_1$, detects the change and passes the change to Master Applet 124A.

At step 1508, via the dedicated socket connection, Master Applet 124A sends WTS server 144 a command together with the change of name field 1202. Since this change is passed to WTS server 144 via the dedicated socket connec-

tion established for web page **1200**, WTS server **144** is able to recognize the origin of the command, web page **1200**, and the name field upon which the change was made.

At step **1510**, WTS server **144** identifies the session created for terminal **104A**.

At step **1512**, WTS server **144** stores the URL and update of name field **1202** into data list **1**.

It should be noted that the operation shown in FIG. **15** can be used to perform data tracking for the other four data fields on web page **1200**, and to perform data tracking for all participant terminals.

Referring to FIG. **16**, there is shown a flowchart illustrating the operation of joining a session by a supervisor, in accordance with the present invention. In the example shown in FIG. **16**, it is assumed that the supervisor is on duty at terminal **104K** in a call center.

As shown in FIG. **16**, at step **1604**, the supervisor performs the steps shown in FIG. **7**, where the supervisor downloads a supervisor page on which a (administration) Master Applet (referred as Master-Applet$_1$) and a Supervisor Applet are imbedded. The Supervisor Applet displays a supervisor session interface (as shown in FIG. **8B**) in a first browser instance (see **800B** in FIG. **17**) on terminal **104K**. Master-Applet$_1$ maintains a dedicated socket connection to WTS gateway **142** for the first browser instance (see **800B** shown in FIG. **17**).

At step **1606**, from the first browser instance (see **800B** shown in FIG. **17**), the supervisor selects a session ID (listed in text box **832**) and then select session button **838**.

At step **1608**, in response to the selection of the select session button **838**, browser **114K** downloads a supervisor agent page, on which a (administration) Master Applet (referred as Master-Applet$_2$) and an Agent Applet are embedded. The Agent Applet creates a supervisor agent session interface **800C** (as shown in FIG. **8C**) and displays it in a second browser instance (see **800C** in FIG. **17**) on terminal **104K**. Master-Applet$_2$ opens and maintains a dedicated socket connection to WTS gateway **142** for the second browser instance (see supervisor agent session interface **800C** shown in FIG. **17**).

At step **1610**, there can be two possible scenarios. A first scenario is that: an agent has joined the selected session to help the consumer, and the supervisor wants to join the selected session as a participant. Under this scenario, the supervisor simply selects join button **846**, and leader and follower buttons **850** and **852** are both selected automatically. A second scenario is that: no agent has joined the session and the supervisor wants to join the session. Under this scenario, the supervisor joins the session just like an agent, by first selecting leader button **850** and/or follower button **852**, and then join session button **846**. In this example, it is assumed that a consumer at terminal **104A** and an agent at terminal **104N** have joined the session, and the supervisor wants to join the selected session.

At step **1612**, the supervisor selects join session button **846**.

At step **1614**, via the socket connection for the second browser instance (see supervisor agent session interface **800C** shown in FIG. **17**), the Master-Applet$_2$ sends WTS server **144** a command together with the selected session ID for the selected session.

At step **1615**, WTS server **144** locates the session indicated by the selected session ID, and sends information stored in participant list **1** and URL history list **1** (see FIG. **6**) to Master-Applet$_2$.

At step **1616**, WTS server **144** stores ParticipantID and ParticipantAddress into participant list **1** for browser **114K**. At this step, participant list **1** includes three participant records (three rows) for browsers **114A**, **114K**, and **114N** respectively.

At step **1618**, Agent Applet at terminal **104K** displays the information stored in participant list 1, and URL history list 1 in participant text box **856**, and URL history text box **858** (see FIG. **8C**) respectively.

At step **1620**, WTS server **144** sends Master-Applet$_2$ the URL of the web page being currently displayed at terminals **104A** and **104N**.

At step **1622**, the Agent Applet at terminal **104K** opens a third browser instance (see **1704** in FIG. **17**).

At step **1624**, browser **114K** downloads the web page identified by the URL from consumer page repository **146** (or loads the web page from memory area **115K** in terminal **104K** if it is cached there), and displays it in the third browser instance (see **1704** in FIG. **17**).

At step **1626**, browser **114K** downloads (consumer) Master Applet **124K**, DTS Applets **126K**, and SessionID Applet **128** from consumer page repository **146** according to the applet tags in the current web page (assuming that these Applets have not previously downloaded).

At step **1628**, Master Applet **124K** opens a dedicated socket and establishes a socket connection to WTS gateway **142** for the third browser instance **1704** shown in FIG. **17**.

After step **1630**, the web pages displayed in third browser instance **1704** at terminal **104K** are being synchronized with the web pages being displayed at terminals **104A** and **104N**.

Referring to FIG. **17**, there is shown three browser instances (**800B**, **800C**, and **1704**) for the supervisor in response to the steps shown in FIG. **16**, in accordance with the present invention.

Referring to FIG. **18**, there is shown a flowchart illustrating the operation of re-browsing a web page previously reviewed in a session, in accordance with the present invention.

In the example shown in FIG. **18**, it is assumed that: (1) a consumer at terminal **104A** is browsing web pages from consumer page repository **146** via browser **114A**, (2) a session list **1** shown in FIG. **6** has been created for browser **114A**, (3) an agent (or a supervisor) is on duty at terminal **104N** in a call center, and agent class (or supervisor class) has been assigned to browser **104N**, (4) at browser **114N**, the first and second browser instances for the agent as shown in FIG. **10** (or the first, second and third browser instances for the supervisor as shown in FIG. **17**) have been displayed, (5) via their respective socket connections established by their Master Applets, the first and second browser instances for the agent as shown in FIG. **10** (or the first, second and third browser instances for the supervisor as shown in FIG. **17**) have been connected to WTS gateway **142**, (6) Master Applets (**124A** and **124N**), DTS Applets (**126A** and **126N**) and SessionID Applets (**128A** and **128N**) have been downloaded into terminals **104A** and **104N** respectively, (7) the agent (or supervisor) has selected and joined the session created for browser **114A**, (8) at browser **114N**, the second browser instance for the agent as shown in FIG. **10** (or the third browser instance for the supervisor as shown in FIG. **17**) is being synchronized with browser **114A**, and (9) bi-direction synchronization has been selected for browsers **114A** and **114N**.

At step **1802**, for an agent user, via scrollable list box **818** on agent session interface shown in FIG. **10**, he/she reviews the URLs for all the web pages previously browsed by browser **114A** in the selected session. For a supervisor, via scrollable list box **858** on supervisor session interface shown in FIG. **17**, he/she reviews the URLs for all the web pages previously browsed by browser **114A** in the selected session.

At step **1804**, to display an individual web page previously browsed by browser **114A**, the agent (or supervisor) selects a URL from scrollable list box **818** (or scrollable list box **858**) and double-clicks on it.

At step **1806**, for the agent, the (agent) Master Applet or (the supervisor Master Applet) sends WTS server **144** a command together with the selected URL, via its respective socket connection.

At **1808**, WTS server **144** sends a command together with the URL and the time information (loading and unloading) to Master Applets **124A** and **124N**, so that Master Applets **124A** and **124N** can inform their respective browsers **114A** and **114N** to load and display the web page based on the URL.

At **1810**, WTS server **144** checks whether browser **114A** previously performed any activities to data fields on the web page identified by the URL, based on the information stored in URL history list **1** and data list **1**. As shown in FIG. **6**, URL history **1** contains the information about: (a) participant ID of browser **114A**, (b) the URL, and (c) the loading and unloading time of the web page identified by the URL. Data list **1** contains the information about: (a) data field names for data fields, (b) value of the data fields, and (c) the times at which values of the data fields were changed.

At step **1812**, if browser **114** previously performed any activities to the data fields on the web page identified by the URL, WTS server **144** sends a command (together with the data field names, values of the data fields, and time information) to Master Applet **124A** (at browser **114A**) and Master Applet **124N** (at browser **114N**).

At step **1814**, at browser **114A**, Master Applet **124A** passes the command, data field names, and data field values to DST Applets **126A**, so that DTS Applets **124A** can display the data field values into respective data fields on the web page being displayed. At browser **114N**, Master Applet **124N** passes the command, data field names, and data field values to DST Applets **126N**, so that DTS Applets **124N** can display the data field values into respective data fields on the web page being displayed.

Since the loading time and unloading time of the URL and the setting time for a data field are recorded in URL history list **1** and data list **1**, if desired, the web page identified by the URL and the activities performed to the data fields can be duplicated (loading the web page, setting data fields on the web page, and unloading the web page) according to the time information.

Referring to FIG. **19**, there is shown a flowchart illustrating the operation of re-browsing all web pages previously reviewed in a session, in accordance with the present invention.

In the example shown in FIG. **19**, it is assumed that: (1) a consumer at terminal **104A** is browsing web pages from consumer page repository **146** via browser **114A**, (2) a session list **1** shown in FIG. **6** has been created for browser **114A**, (3) an agent (or a supervisor) is on duty at terminal **104N** in a call center, and agent class (or supervisor class) has been assigned to browser **104N**, (4) at browser **114N**, the first and second browser instances for the agent as shown in FIG. **10** (or the first, second and third browser instances for the supervisor as shown in FIG. **17**) have been displayed, (5) via their respective socket connections established by their respective Master Applets, the first and second browser instances for the agent as shown in FIG. **10** (or the first, second and third browser instances for the supervisor as shown in FIG. **17**) have been connected to WTS gateway **142**, (6) Master Applets (**124A** and **124N**), DTS Applets (**126A** and **126N**) and SessionID Applets (**128A** and **128N**) have been downloaded into terminals **104A** and **104N** respectively, (7) the agent (or supervisor) has selected and joined the session created for browser **114A**, (8) at browser **114N**, the second browser instance for the agent as shown in FIG. **10** (or the third browser instance for the supervisor as shown in FIG. **17**) is being synchronized with browser **114A**, and (9) bi-direction synchronization has been selected for browsers **114A** and **114N**.

At step **1902**, for an agent user, via scrollable list box **818** on agent session interface shown in FIG. **10**, he/she reviews the URLs for all the web pages previously browsed by browser **114A** in the selected session. For a supervisor, via scrollable list box **858** on supervisor session interface shown in FIG. **17**, he/she reviews the URLs for all the web pages previously browsed by browser **114A** in the selected session.

At step **1904**, to display all web pages previously browsed by browser **114A**, the agent (or supervisor) selects Go to URLs button **820** in the agent session interface as shown in FIG. **10** (or Go to URLs button **860** in the supervisor session interface as shown in FIG. **17**.

At step **1906**, the (agent) Master Applet, or the (supervisor) Master Applet, sends a command to WTS server **144**.

At **1908**, WTS server **144** sequentially sends commands, together the URLs and time information, to Master Applets **124A** and **124N**, so that Master Applets **124A** and **124N** can inform their respective browsers **114A** and **114N** to load and display the web pages based on the URLs.

At **1910**, for each one of URLs that are sent together with the commands, WTS server **144** checks whether browser **114A** previously performed any activities to data fields on a web page identified by a respective URL, based on the information stored in URL history list **1** and data list **1**.

At step **1912**, if browser **114** previously performed any activities to the data fields on the web page identified by a respective URL, WTS server **144** sends a command (together with the data field names and values of the data fields) to Master Applets **124A** (at browser **114A**) and Master Applet **124N** (at browser **114N**).

At step **1914**, at browser **114A**, Master Applet **124A** passes the command, data field names, and data field values to DST Applets **126A**, so that DTS Applets **124A** can display the data field values into respective data fields on the web page being currently displayed. At browser **114N**, Master Applet **124N** passes the command, data field names, and data field values to DST Applets **126N**, so that DTS Applets **124N** can display the data field values into respective data fields on the web page being currently displayed.

Since the loading time and unloading time of the URLs and the setting time for data fields are recorded in URL history list **1** and data list **1**, if desired, all the web pages identified by the URLs and the activities performed to the data fields can be duplicated (loading the web page, setting data fields on the web page, and unloading the web page) according to the timing information.

It should be noted that, in the above-described embodiments, all the Applets (Master Applets, DTS Applets, SessionID Applets, and Agent Applet) embedded into web pages are written using Java. However, some or all of these Applets can be written using a browser script language, such as Java Script. More specifically, the codes for these Applets can be selectively written into web pages using the browser script language, instead of using applet tags to link these Applets. When a web browser downloads a web page containing the Applets written in browser script language, it stores these Applets into the memory area of the terminal on which the web browser is running, and then initializes and invokes them.

The present invention has the following advantages:

Dependable web page tracking and synchronizing—It tracks and synchronizes all user activities, even if web pages come from cached pages stored in browser cache or proxy servers.

Ease of use—It eliminates the current manual process of multiple users separately re-creating the web navigation.

Ease of execution (from users' point of view)—It does not require additional software to support the present invention. No software needs to be installed, configured, or run by a user.

Portability—It works across different operating systems at both client and server sides. On the client side, the requirement is that there be a web browser that supports Java Applets. On the server side, the requirement is that there be a Java Virtual Machine (JVM) on the same server that provides the HTTP service. Since there are JVMs practically for very operating system, the server components of the present invention have the potential to run on all the operating systems.

Compatibility—It works together with any HTTP servers from different vendors because the server components of the present invention requires no processing by HTTP servers, and thus are independent from HTTP servers.

Flexibility—Web page synchronization can be used independently in conjunction with web page tracking. Web page synchronization does not require persistent storage of any of the data tracked.

User privacy—It ensures a reasonable level of user privacy, since tracking and synchronization is limited to pages served by a web site that the information provider has control over.

Multiple HTTP server supported—It can handle the situation where a company has multiple physical servers running its web site, since the separation of the WTS gateway and server components enables a gateway to be placed on each HTTP server—each communicating with a common WTS server.

While the invention has been illustrated and described in detail in the drawing and foregoing description, it should be understood that the invention may be implemented through alternative embodiments within the spirit of the present invention. Thus, the scope of the invention is not intended to be limited to the illustration and description in this specification, but is to be defined by the appended claims.

What is claimed is:

1. A method for tracking interactions with pages that have been loaded from a web server to a terminal during a user session, and for storing information about the interactions to a sage tracking server, comprising the steps of:

loading a first page from the web server, the first page being associated with a page locator for indicating a location of the first page in the web server, and the first page containing location information for indicating a location of a program;

loading the program from the web server based on the location information, and executing the program;

the program monitoring interactions with the page; and

the program sending information about the interactions to the page tracking server during the session,

creating a session table using sent information about the interactions,

creating a sessionID for the session table wherein each sessionID is associated with a session list for maintaining information about a session, a participant list for maintaining information about all participant browsers in a session, a URL history list for maintaining information about all web pages visited by all participants in a session, a data list for maintaining information about the data fields on the web pages visited by all participants in a session, and a command list for maintaining information about all commands issues to the server by the various participants in a session,

wherein the data list includes data fields for a Session ID for linking the data list to a session, a WasRelayed for indicating if this data field has been broadcasted, a FieldName for indicating the actual name of the data field, a DataName for indicating the name of the data field displayed on a web page, a DataValue for indicating the value of the data field, a TimeStamp for indicating the time at which this data field is updated, a URL for indicating the web page on which the data field was displayed, and a ParticipantID for indicating the participant browser who updated this data field.

2. The method of claim 1, further comprising the steps of:

loading a second page from the web server, the second page being associated with a page locator for indicating a location of the second page in the web server, the second page containing location information for indicating a location of the program loaded in said loading step; and

executing the program based on the location information in the second page;

the program monitoring interactions with another page; and

the program sending the information about the interactions with the second page to the page tracking server during the session.

3. The method of claim 2, further comprising the step of:

storing the information about the interactions with the second page in the page tracking server during the session.

4. The method of claim 2, comprising storing the sent information from the web servers.

5. The method of claim 2, comprising collecting and analyzing the stored information about the interactions.

6. The method of claim 1, further comprising the step of:

storing the information about the interactions with the first page in the page tracking server.

7. The method of claim 1, the page locator being URL, and the location information being a tag.

8. The method of claim 1, the web server being an HTTP server.

9. The method of claim 1, comprising opening a dedicated socket between the terminal and a gateway.

10. The method of claim 1, comprising informing the page tracking server that the first page has been loaded on the terminal.

11. The method of claim 1, wherein a session is defined as a collection web page interactions that occur over a given period of time from a specific browser.

12. The method of claim 1, further comprising creating a session table using sent information about the interactions.

13. The method of claim 12, comprising creating a sessionID for the session table wherein each sessionID is associated with a session list for maintaining information about a session, a participant list for maintaining information about all participant browsers in a session, a URL history list for maintaining information about all web pages visited by all participants in a session, a data list for maintaining information about the data fields on the web

pages visited by all participants in a session, and a command list for maintaining information about all commands issues to the server by the various participants in a session.

14. The method of claim 13, wherein the session list includes a sessionID for identifying a session, a user name for indicating the actual name for whom the session is created, a start time for indicating the time of starting the session, a stop time for indicating the time of stopping the session, and session notes for recording the notes of the session.

15. The method of claim 14, wherein the participant list includes a sessionID for linking the participant list to a session, a participantID for identifying a participant, ParticipantAddresses for indicating a participant's IP address, a class for indicating a user class of the participant, and a direction for indicating the synchronization direction for the participant browser.

16. The method of claim 13, wherein the URL history list includes a SessionID for linking the URL history list to a session, a PageURL for indicating the URL of a web page visited, a ParticipantID for identifying a participant who visited the web page, a loading time for indicating the loading time of the web page, and an unloading time for indicating the unloading time of the web page.

17. The method of claim 13, wherein the command list includes data fields for a Session ID for linking the data list to session, a Command for indicating the specific command executed, a URL for indicating the web page to which the command operated, a FieldPoint for indicating the data field to which the command operated, and a TimeStamp for indicating the time at which command was executed.

18. The method of claim 1, further comprising creating a session interface for an administrator to monitor the information about the interactions.

19. The method of claim 18, comprising joining the administrator to a session to monitor the information about the interactions.

20. The method of claim 1, further comprising creating a browser supervisor session interface to monitor the information about the interactions.

21. A method for tracking interactions with pages that have been loaded from a web server to a terminal during a user session, and for storing information about the interactions to a page tracking server, comprising the steps of:

loading a first page from the web server, the first page being associated with a page locator for indicating a location of the first page in the web server, and the first page containing location information for indicating a location of a program;

loading the program from the web server based on the location information, and executing the program;

the program monitoring interactions with the page; and

the program sending information about the interactions to the page tracking server during the session,

joining the administrator to a session to monitor the information about the interactions,

wherein the session interface includes a text box for entering a SessionID, a join session button for joining a session identified by the Session ID, a drop button for leaving a session, a leader check box, a follower check box, a scrollable list box for displaying the information contained in the participant list associated with a selected session, a scrollable list box for displaying the information in an identified URL history list, and a text box for displaying the information in an identified data list.

22. The method of claim 21, further comprising the steps of:

loading a second page from the web server, the second page being associated with a page locator for indicating a location of the second page in the web server, the second page containing location information for indicating a location of the program loaded in said loading step; and

executing the program based on the location information in the second page;

the program monitoring interactions with another page; and

the program sending the information about the interactions with the second page to the page tracking server during the session.

23. The method of claim 21, further comprising the step of:

storing the information about the interactions with the first page in the page tracking server.

24. The method of claim 21, the page locator being URL, and the location information being a tag.

25. The method of claim 21, the web server being an HTTP server.

26. The method of claim 21, comprising opening a dedicated socket between the terminal and a gateway.

27. The method of claim 21, comprising informing the page tracking server that the first page has been loaded on the terminal.

28. The method of claim 21, wherein a session is defined as a collection of web page interactions that occur over a given period of time from a specific browser.

29. A method for tracking interactions with pages that have been loaded from a web server to a terminal during a user session, and for storing information about the interactions to a page tracking server, comprising the steps of:

loading a first page from the web server, the first page being associated with a page locator for indicating a location of the first page in the web server, and the first page containing location information for indicating a location of a program;

loading the program from the web server based on the location information, and executing the program;

the program monitoring interactions with the page; and

the program sending information about the interactions to the page tracking server during the session,

creating a browser supervisor session interface to monitor the information about the interactions,

wherein the supervisor session interface includes a scrollable list box for displaying session IDs of all active sessions in the session table and for selecting one of the session IDs, a text box for displaying relevant statistics of the server, a multi-column scrollable list box for displaying details about the session selected in the scrollable list box, a select session button for selecting a session from the scrollable list box.

30. The method of claim 29, comprising monitoring operational status of a session using the scrollable list box.

31. The method of claim 29, further comprising the steps of:

loading a second page from the web server, the second page being associated with a page locator for indicating a location of the second page in the web server, the second page containing location information for indicating a location of the program loaded in said loading step; and

executing the program based on the location information in the second page;

the program monitoring interactions with another page; and

the program sending the information about the interactions with the second page to the page tracking server during the session.

32. The method of claim 29, further comprising the step of:

storing the information about the interactions with the first page in the page tracking server.

33. The method of claim 29, the page locator being URL, and the location information being a tag.

34. The method of claim 29, the web server being an HTTP server.

35. The method of claim 29, comprising opening a dedicated socket between the terminal and a gateway.

36. The method of claim 29, comprising informing the page tracking server that the first page has been loaded on the terminal.

37. The method of claim 29, wherein a session is defined as collection of web page interactions that occur over a given period of time from a specific browser.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,035,332

DATED : March 7, 2000

INVENTOR(S) : Michael I. Ingrassia, Jr., et al.

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 21, line 50, delete "sage" and substitute –page--.

Signed and Sealed this

Thirteenth Day of March, 2001

Attest:

NICHOLAS P. GODICI

Attesting Officer

Acting Director of the United States Patent and Trademark Office

US006418471B1

(12) **United States Patent**
Shelton et al.

(10) **Patent No.: US 6,418,471 B1**
(45) **Date of Patent: Jul. 9, 2002**

(54) **METHOD FOR RECORDING AND REPRODUCING THE BROWSING ACTIVITIES OF AN INDIVIDUAL WEB BROWSER**

(75) Inventors: **James A. Shelton**, Holmdel, NJ (US); **Michael I. Ingrassia, Jr.**, Dallas, TX (US)

(73) Assignee: **NCR Corporation**, Dayton, OH (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/151,907**

(22) Filed: **Sep. 11, 1998**

Related U.S. Application Data

(63) Continuation-in-part of application No. 08/944,124, filed on Oct. 6, 1997, now Pat. No. 5,951,643.

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **709/227**
(58) **Field of Search** 709/224, 227, 709/245, 204, 205, 217, 218; 707/5, 104

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,761,436	A *	6/1998	Nielsen	709/245
5,809,250	A *	9/1998	Kisor	709/227
5,845,290	A *	12/1998	Yoshii	707/104
5,951,643	A *	9/1999	Shelton et al.	709/227
5,960,429	A *	9/1999	Peercy et al.	707/5
6,035,332	A *	3/2000	Ingrassia, Jr. et al.	709/224
6,052,730	A *	4/2000	Felciano et al.	709/225

* cited by examiner

Primary Examiner—Robert B. Harrell
(74) *Attorney, Agent, or Firm*—James M. Stover

(57) **ABSTRACT**

A mechanism for recording browser activities performed at a first web browser and subsequently replaying the recorded browser activities at one or more web browsers. The browsing activities can be recorded together with time references corresponding to various browsing activities such as when a web page is retrieved, or the length of time a page is displayed. The browsing activities can be replayed in accordance with the recorded time references. If desired, the browsing activities that are being repeated can be synchronized among multiple browsers.

5 Claims, 20 Drawing Sheets

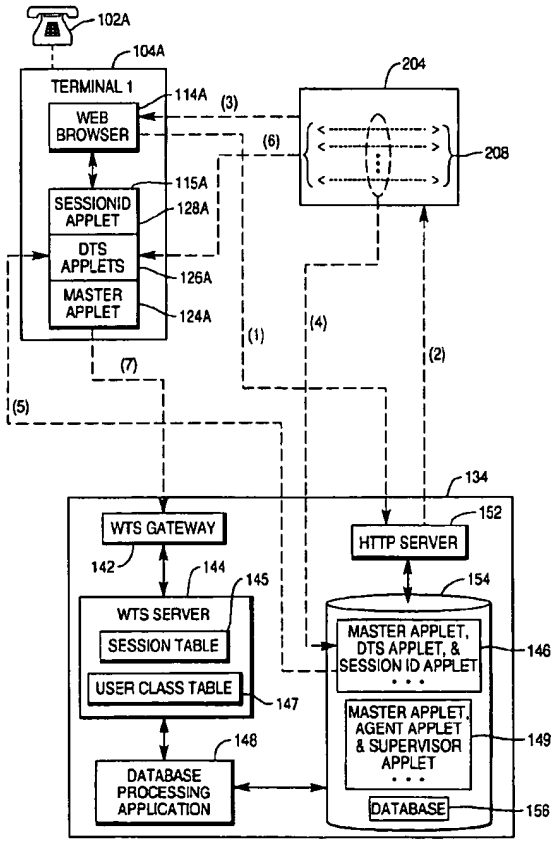
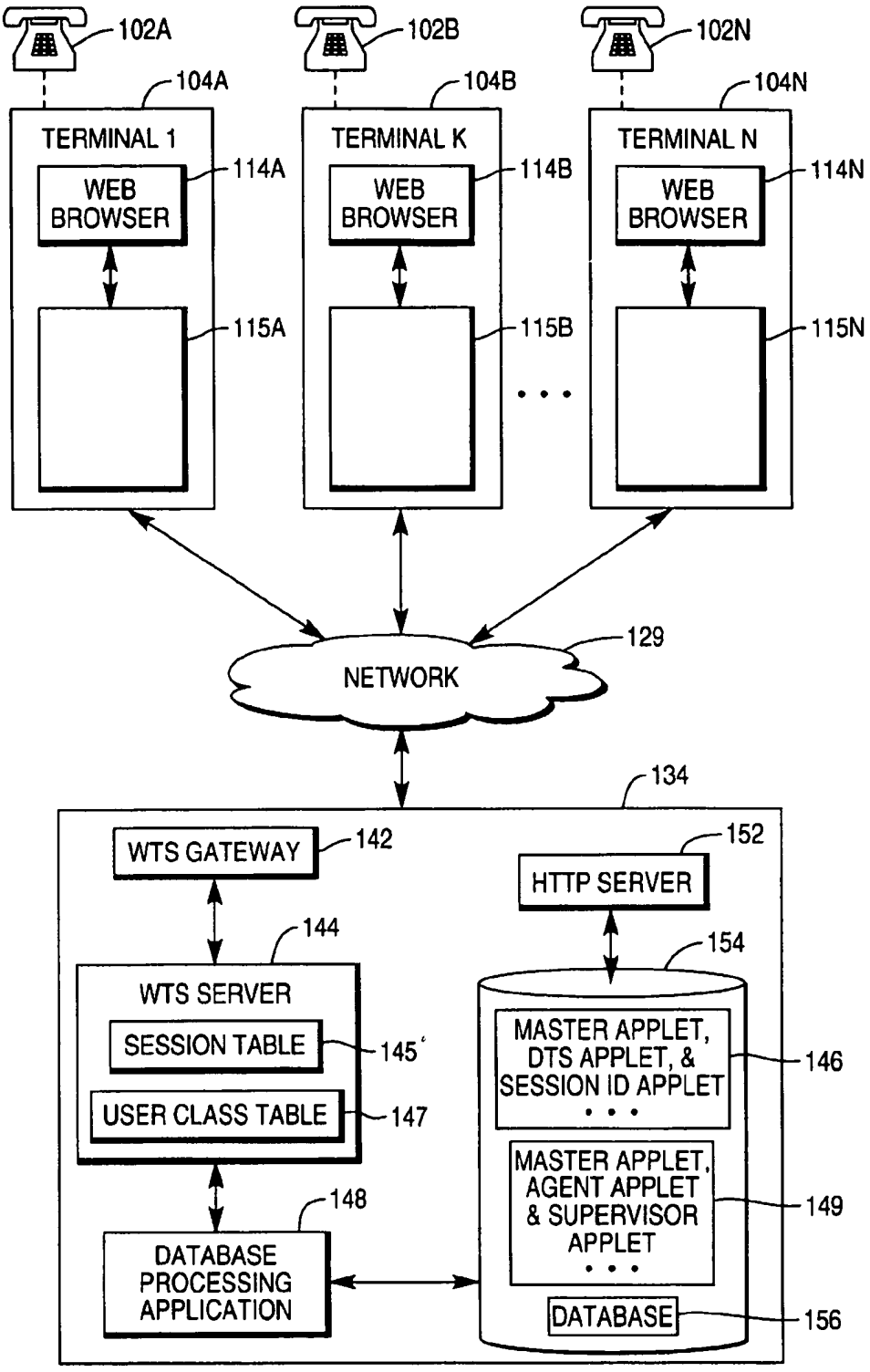


FIG. 1


102A
102B
102N
104A
104B
104N
TERMINAL 1
TERMINAL K
TERMINAL N
WEB BROWSER
WEB BROWSER
WEB BROWSER
114A
114B
114N
115A
115B
115N
129
NETWORK
134
WTS GATEWAY
142
HTTP SERVER
152
144
154
WTS SERVER
SESSION TABLE
145
USER CLASS TABLE
147
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

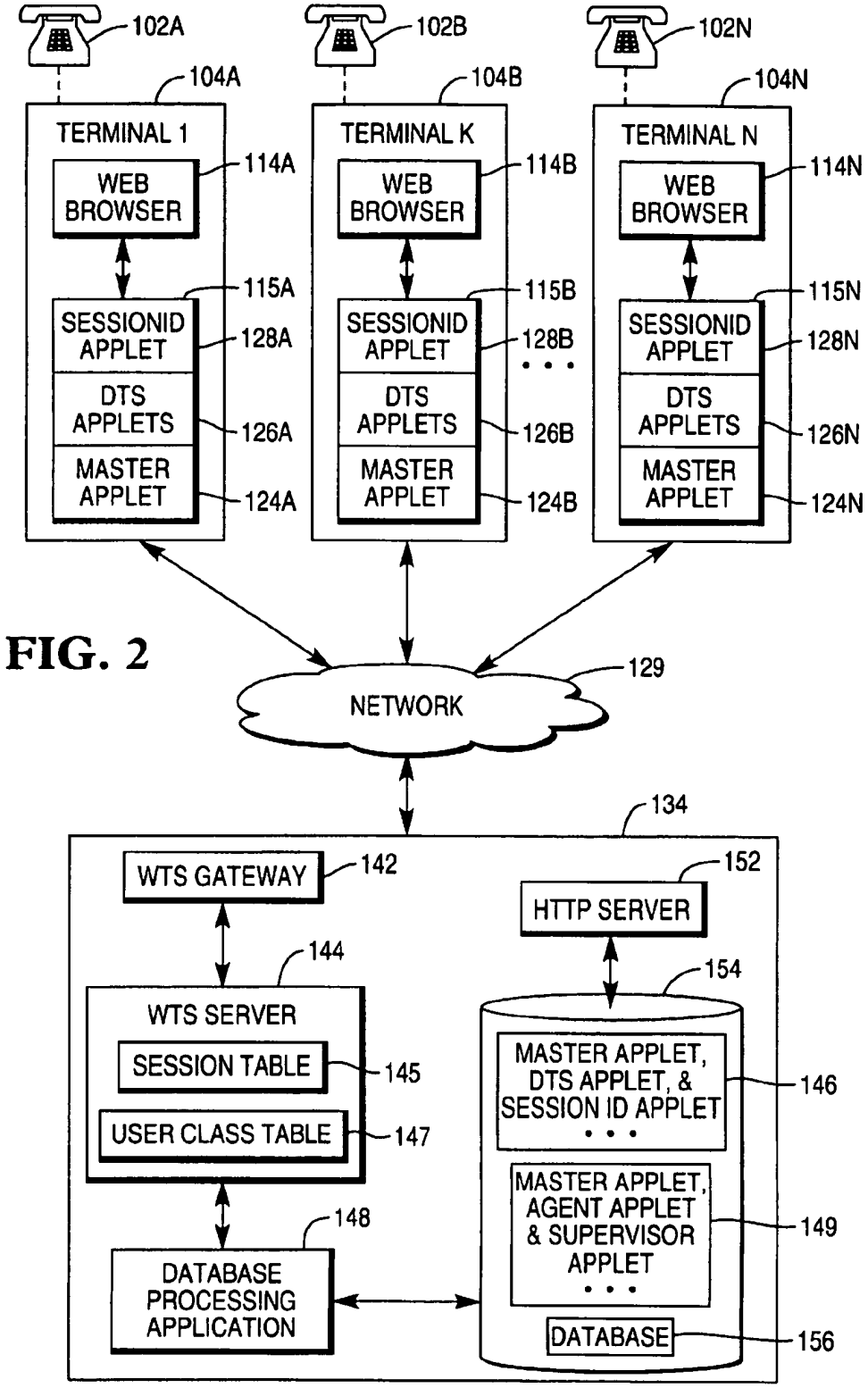

102A
102B
102N
104A
104B
104N
TERMINAL 1
TERMINAL K
TERMINAL N
WEB BROWSER
114A
114B
114N
SESSIONID APPLET
115A
115B
115N
128A
128B
128N
DTS APPLETS
126A
126B
126N
MASTER APPLET
124A
124B
124N
NETWORK
129
134
WTS GATEWAY
142
HTTP SERVER
152
144
154
WTS SERVER
SESSION TABLE
145
USER CLASS TABLE
147
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

FIG. 2

FIG. 3

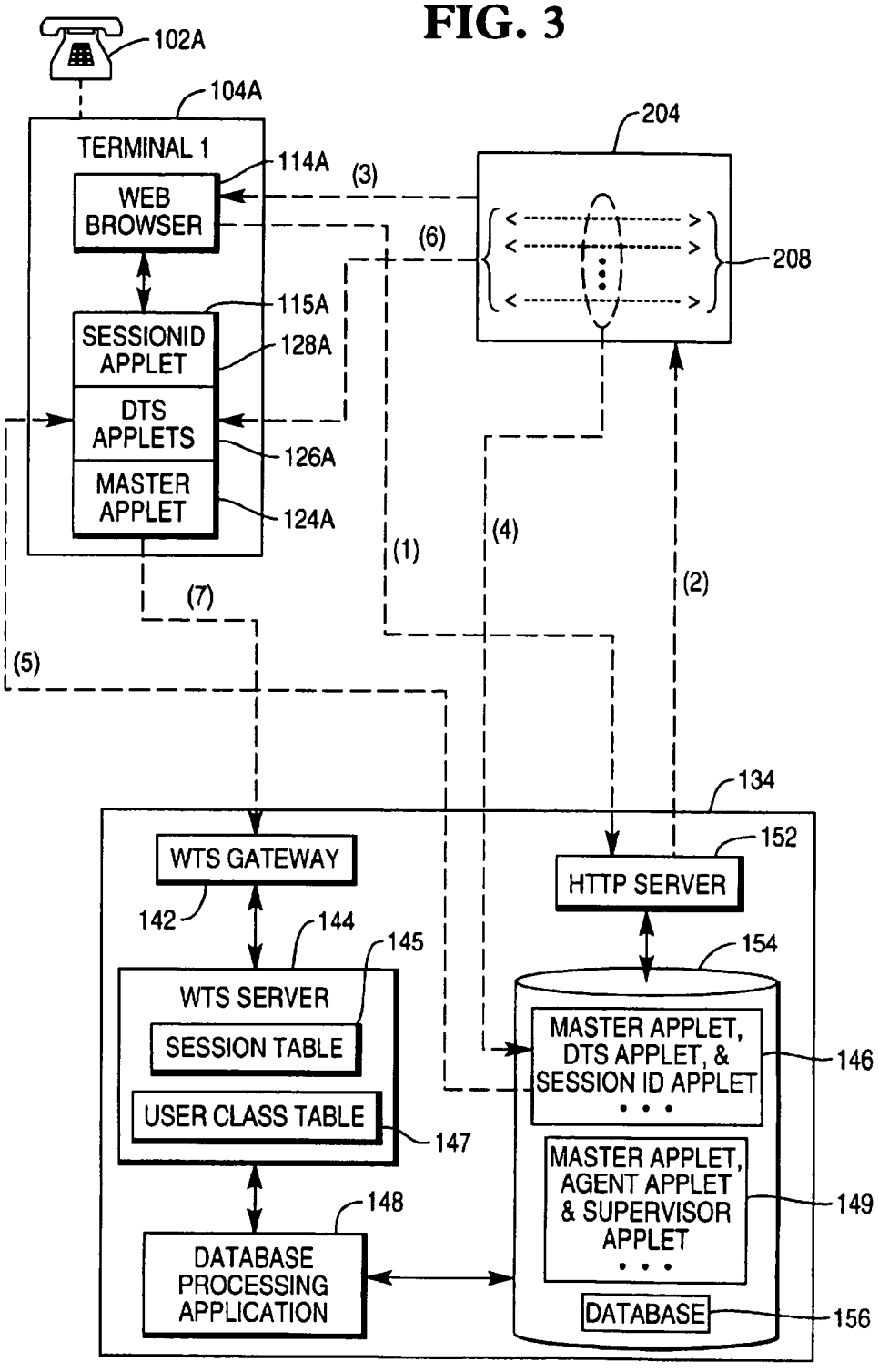

102A
104A
TERMINAL 1
114A
WEB BROWSER
(3)
204
(6)
208
115A
SESSIONID APPLET
128A
DTS APPLETS
126A
MASTER APPLET
124A
(1)
(4)
(2)
(7)
(5)
134
WTS GATEWAY
142
144
145
152
HTTP SERVER
154
WTS SERVER
SESSION TABLE
USER CLASS TABLE
147
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

FIG. 4

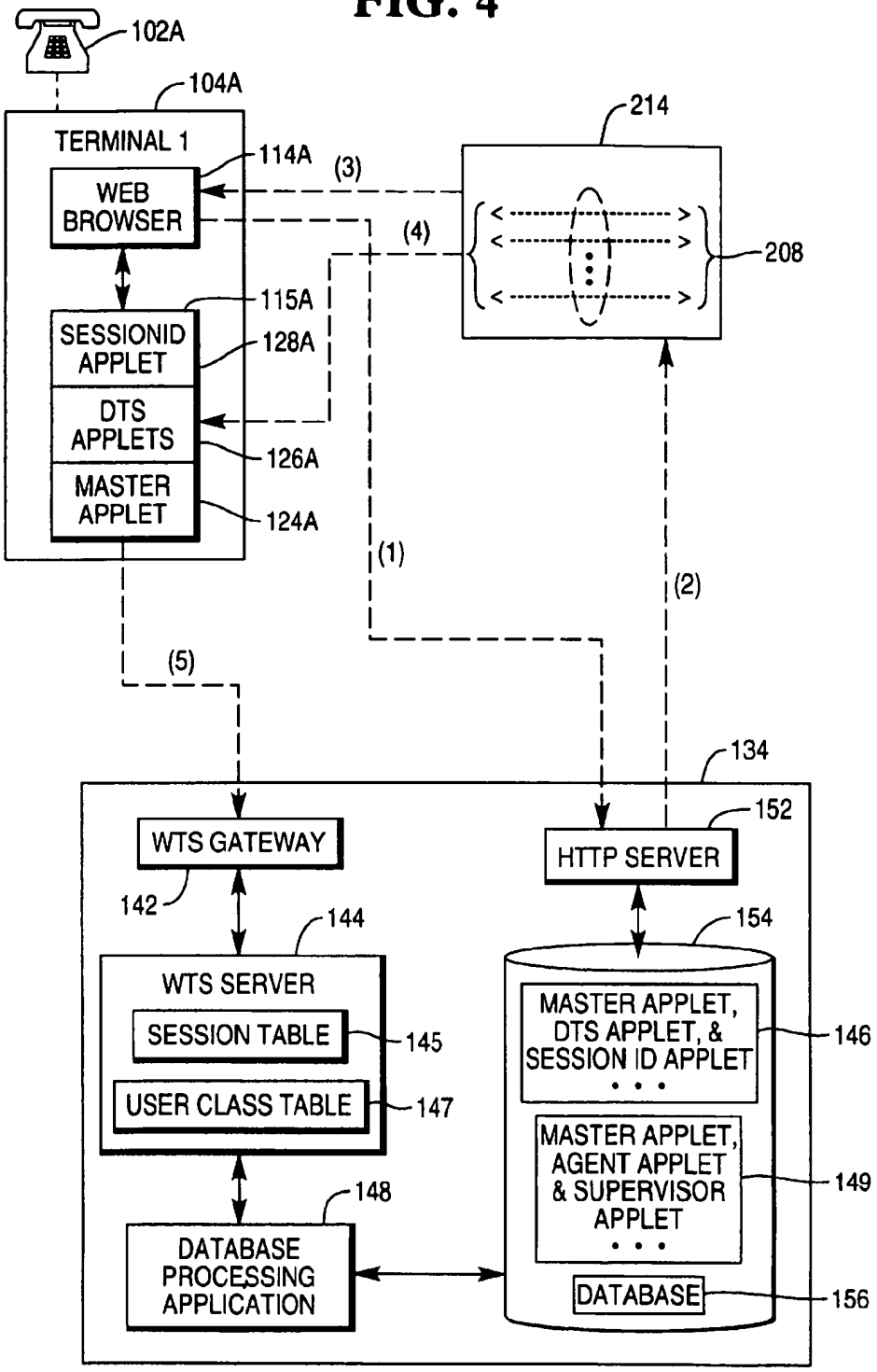

102A
104A
TERMINAL 1
114A
WEB BROWSER
(3)
(4)
214
208
115A
SESSIONID APPLET
128A
DTS APPLETS
126A
MASTER APPLET
124A
(1)
(2)
(5)
134
152
WTS GATEWAY
HTTP SERVER
142
144
154
WTS SERVER
SESSION TABLE
145
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
USER CLASS TABLE
147
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

FIG. 5

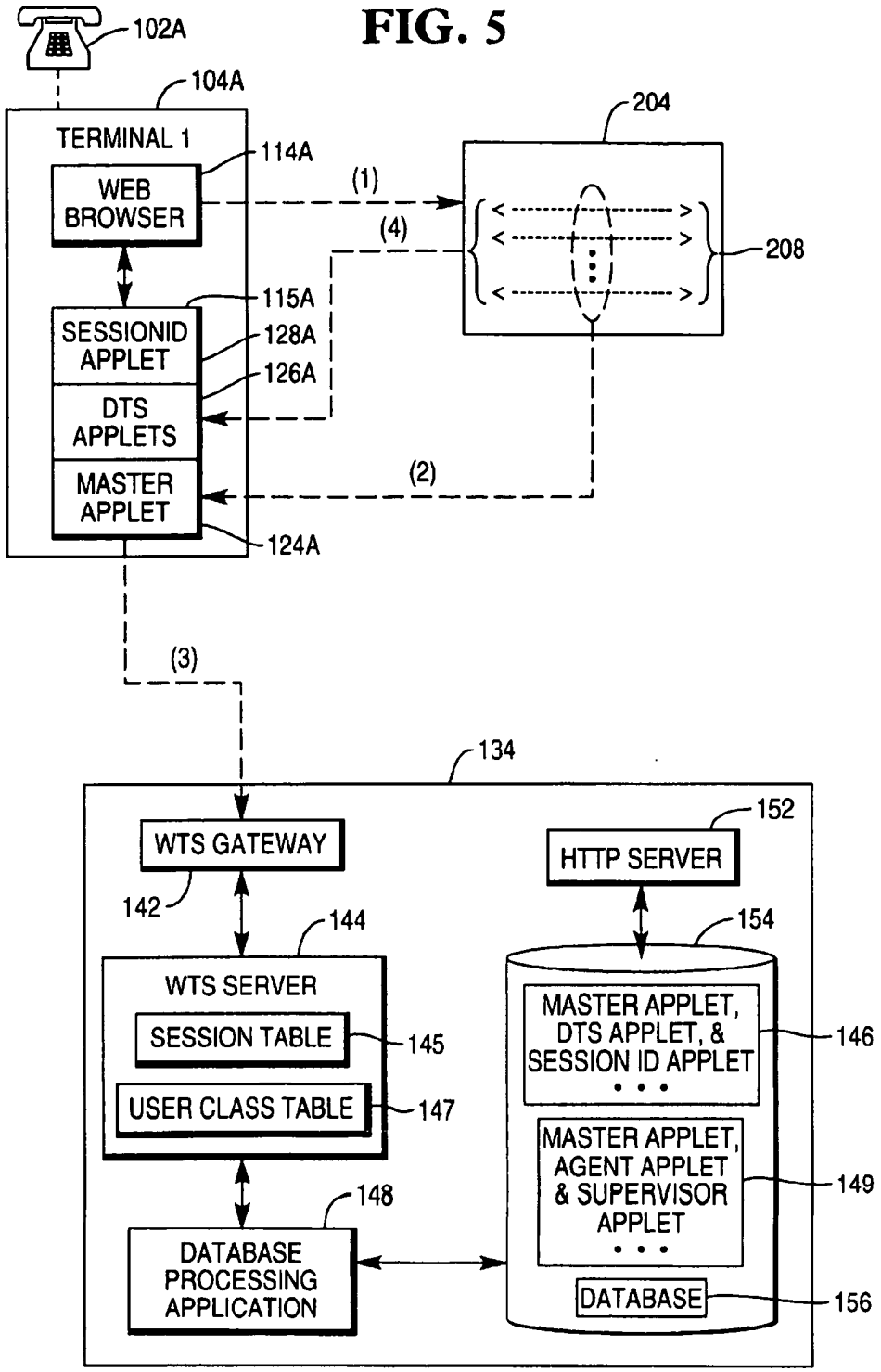

102A
104A
TERMINAL 1
114A
WEB BROWSER
(1)
204
(4)
208
115A
SESSIONID APPLET
128A
126A
DTS APPLETS
MASTER APPLET
(2)
124A
(3)
134
WTS GATEWAY
142
152
HTTP SERVER
144
154
WTS SERVER
SESSION TABLE
145
MASTER APPLET, DTS APPLET, & SESSION ID APPLET
146
USER CLASS TABLE
147
MASTER APPLET, AGENT APPLET & SUPERVISOR APPLET
149
148
DATABASE PROCESSING APPLICATION
DATABASE
156

FIG. 6

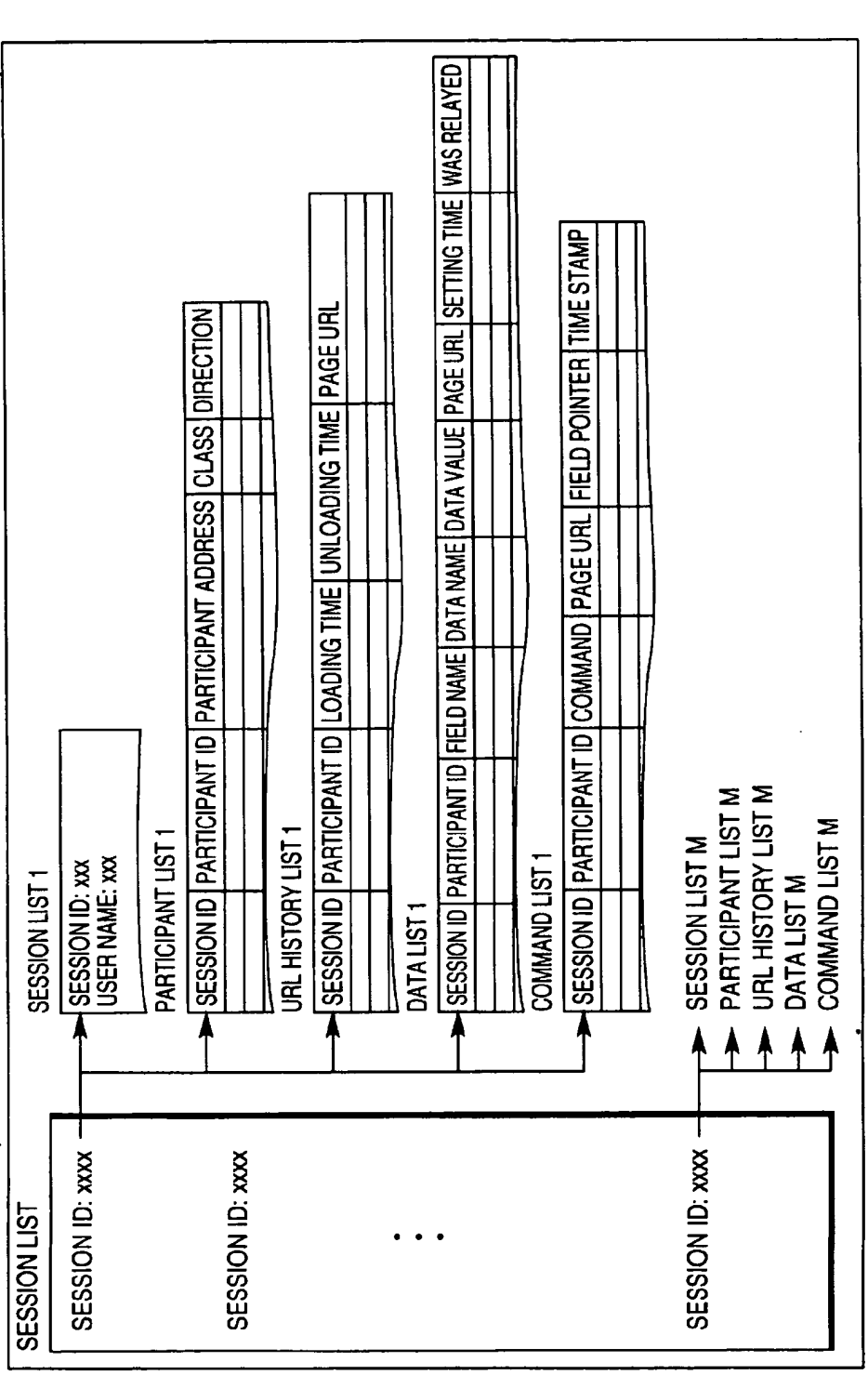
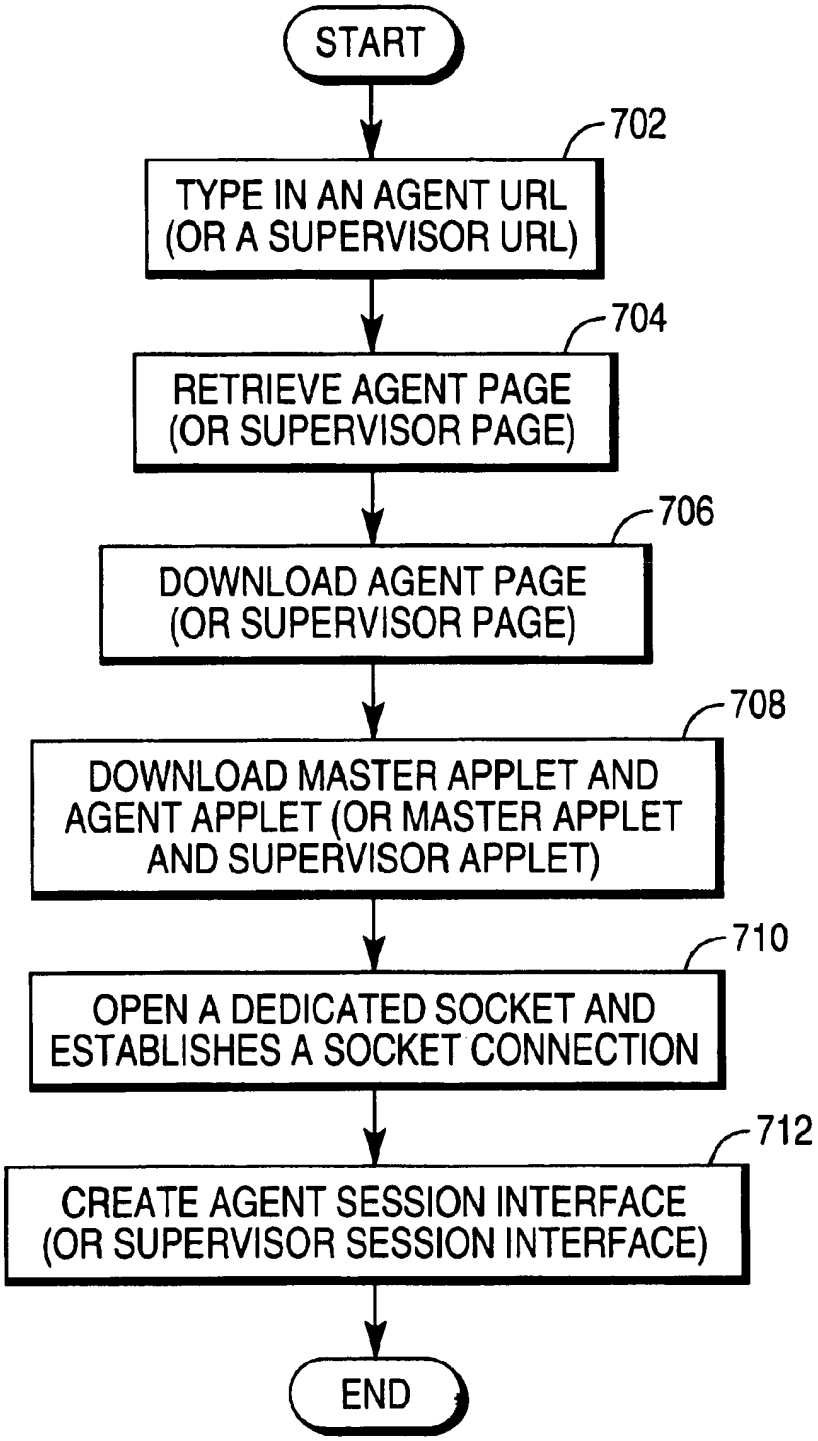

145
SESSION LIST
SESSION ID: xxxx
SESSION ID: xxxx
SESSION ID: xxxx
SESSION LIST 1
SESSION ID: xxx
USER NAME: xxx
PARTICIPANT LIST 1
SESSION ID
PARTICIPANT ID
PARTICIPANT ADDRESS
CLASS
DIRECTION
URL HISTORY LIST 1
SESSION ID
PARTICIPANT ID
LOADING TIME
UNLOADING TIME
PAGE URL
DATA LIST 1
SESSION ID
PARTICIPANT ID
FIELD NAME
DATA NAME
DATA VALUE
PAGE URL
SETTING TIME
WAS RELAYED
COMMAND LIST 1
SESSION ID
PARTICIPANT ID
COMMAND
PAGE URL
FIELD POINTER
TIME STAMP
SESSION LIST M
PARTICIPANT LIST M
URL HISTORY LIST M
DATA LIST M
COMMAND LIST M

FIG. 7


START
702
TYPE IN AN AGENT URL (OR A SUPERVISOR URL)
704
RETRIEVE AGENT PAGE (OR SUPERVISOR PAGE)
706
DOWNLOAD AGENT PAGE (OR SUPERVISOR PAGE)
708
DOWNLOAD MASTER APPLET AND AGENT APPLET (OR MASTER APPLET AND SUPERVISOR APPLET)
710
OPEN A DEDICATED SOCKET AND ESTABLISHES A SOCKET CONNECTION
712
CREATE AGENT SESSION INTERFACE (OR SUPERVISOR SESSION INTERFACE)
END

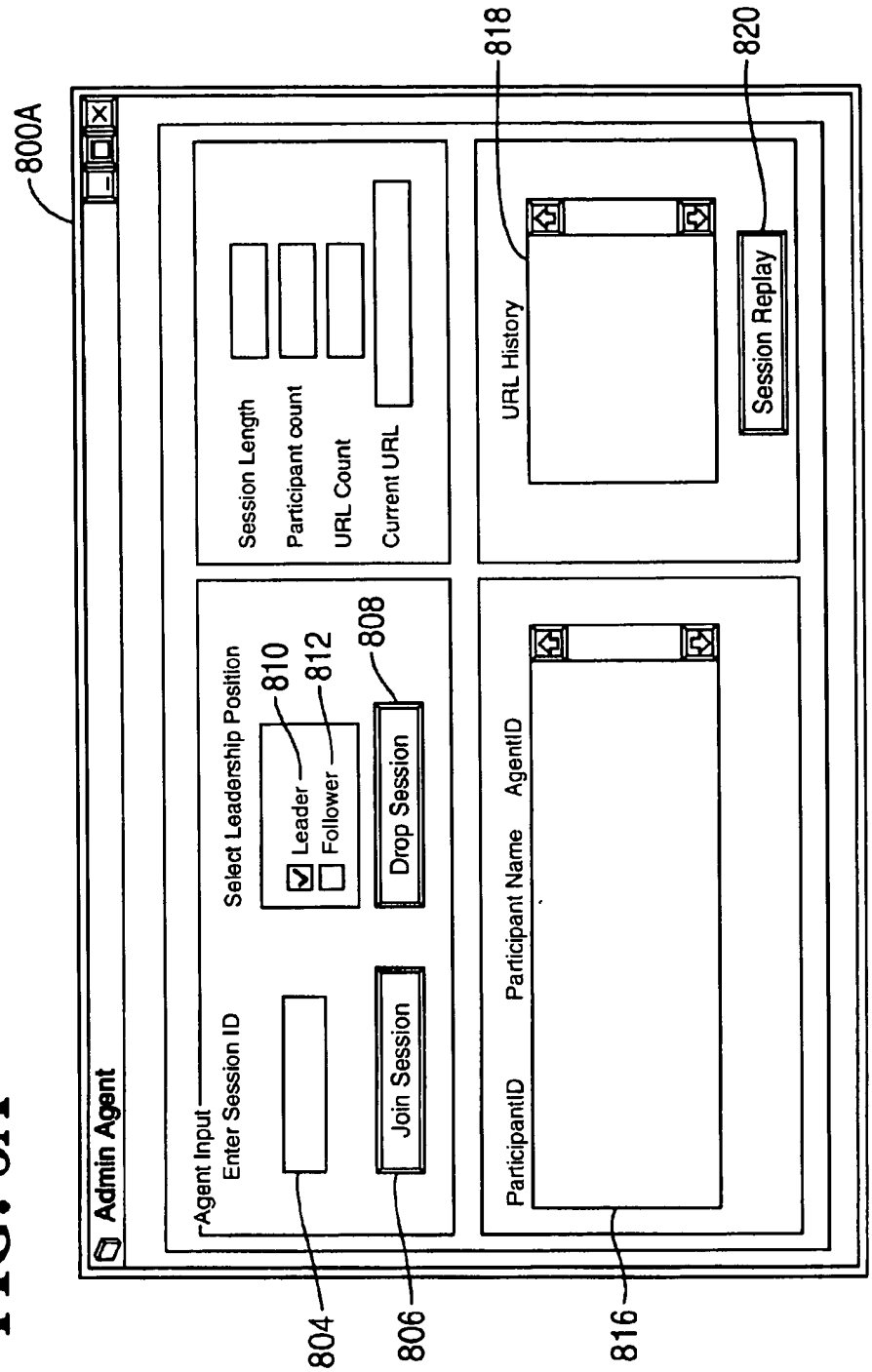
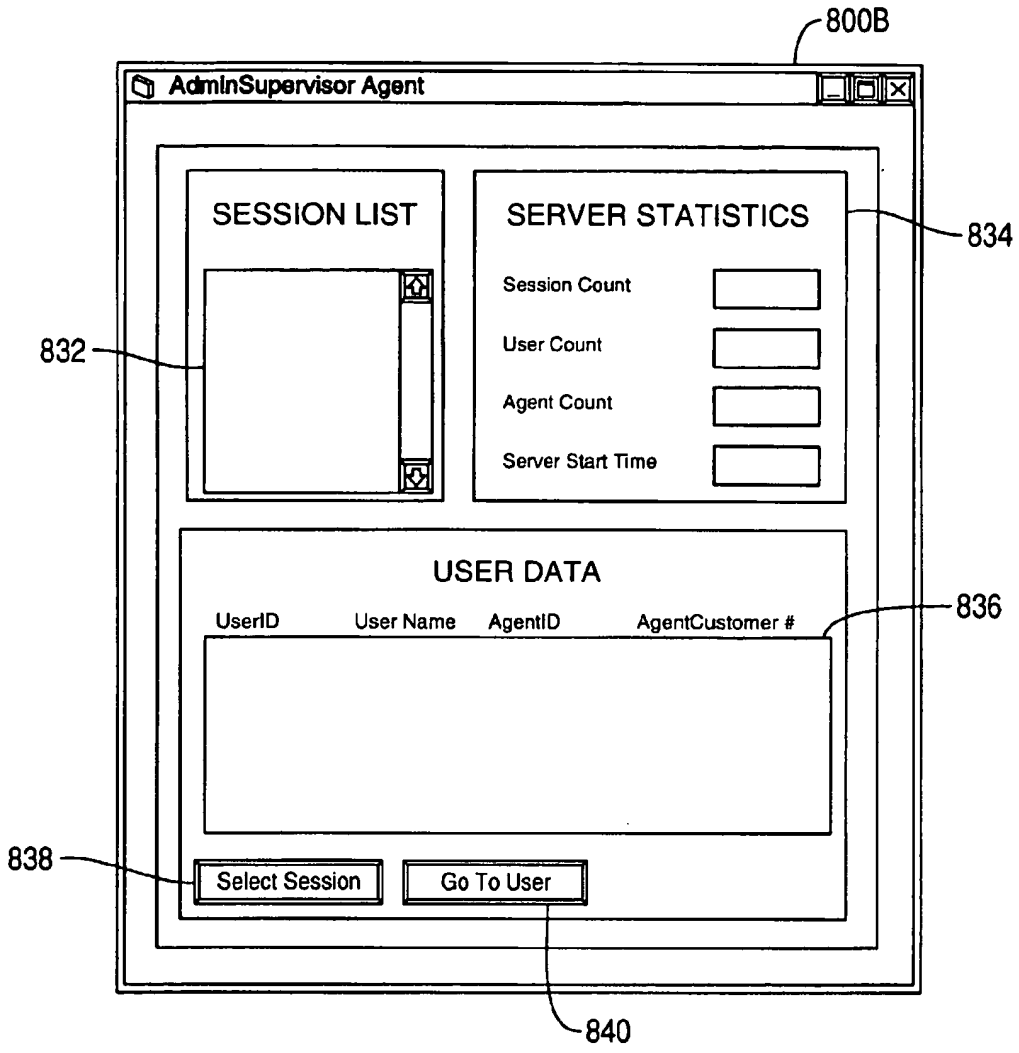

800A
818
820
Admin Agent
Session Length
Participant count
URL Count
Current URL
URL History
Session Replay
Agent Input
Enter Session ID
Select Leadership Position
Leader
Follower
810
812
808
Drop Session
Join Session
ParticipantID
Participant Name
AgentID
804
806
816

FIG. 8A

FIG. 8B


800B
AdminSupervisor Agent
SESSION LIST
SERVER STATISTICS
Session Count
User Count
Agent Count
Server Start Time
834
832
USER DATA
UserID
User Name
AgentID
AgentCustomer #
836
838
Select Session
Go To User
840

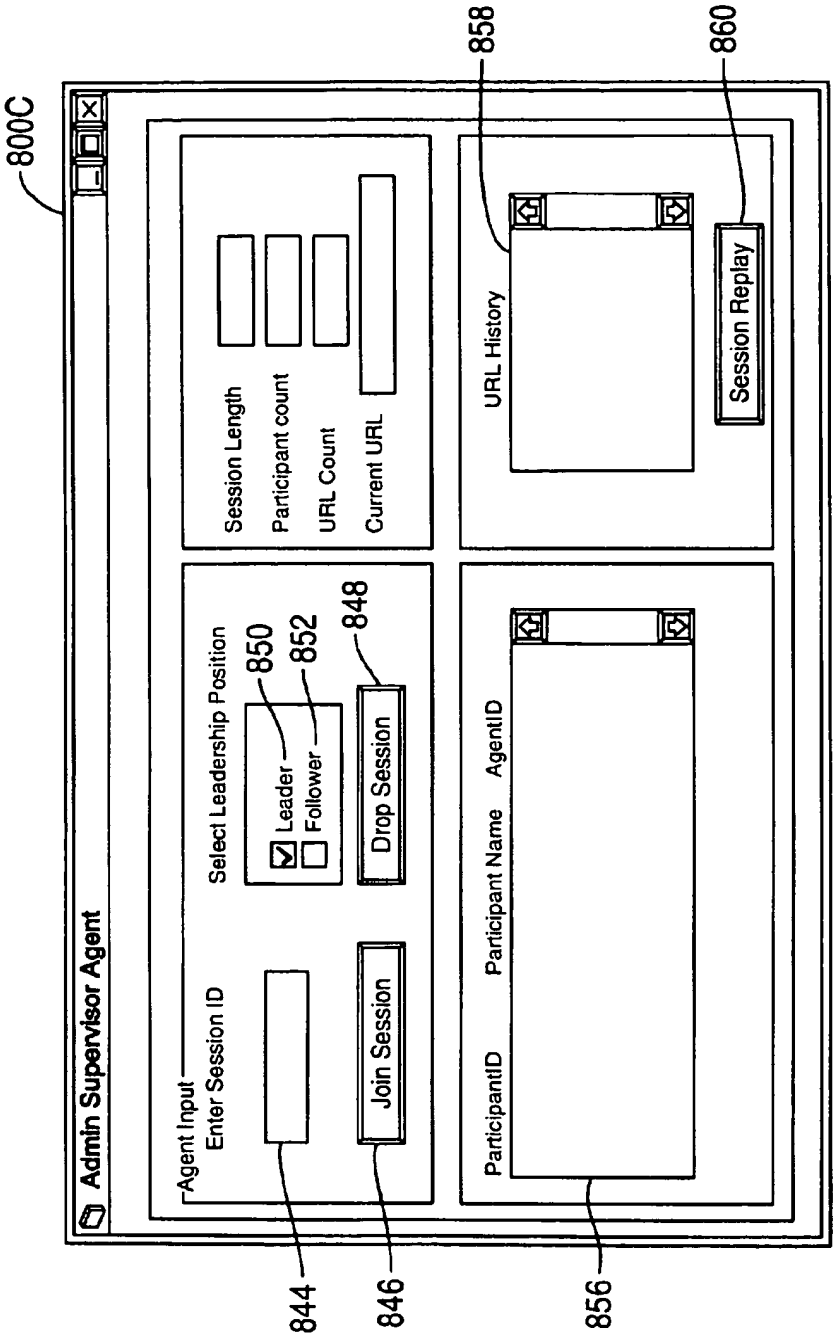
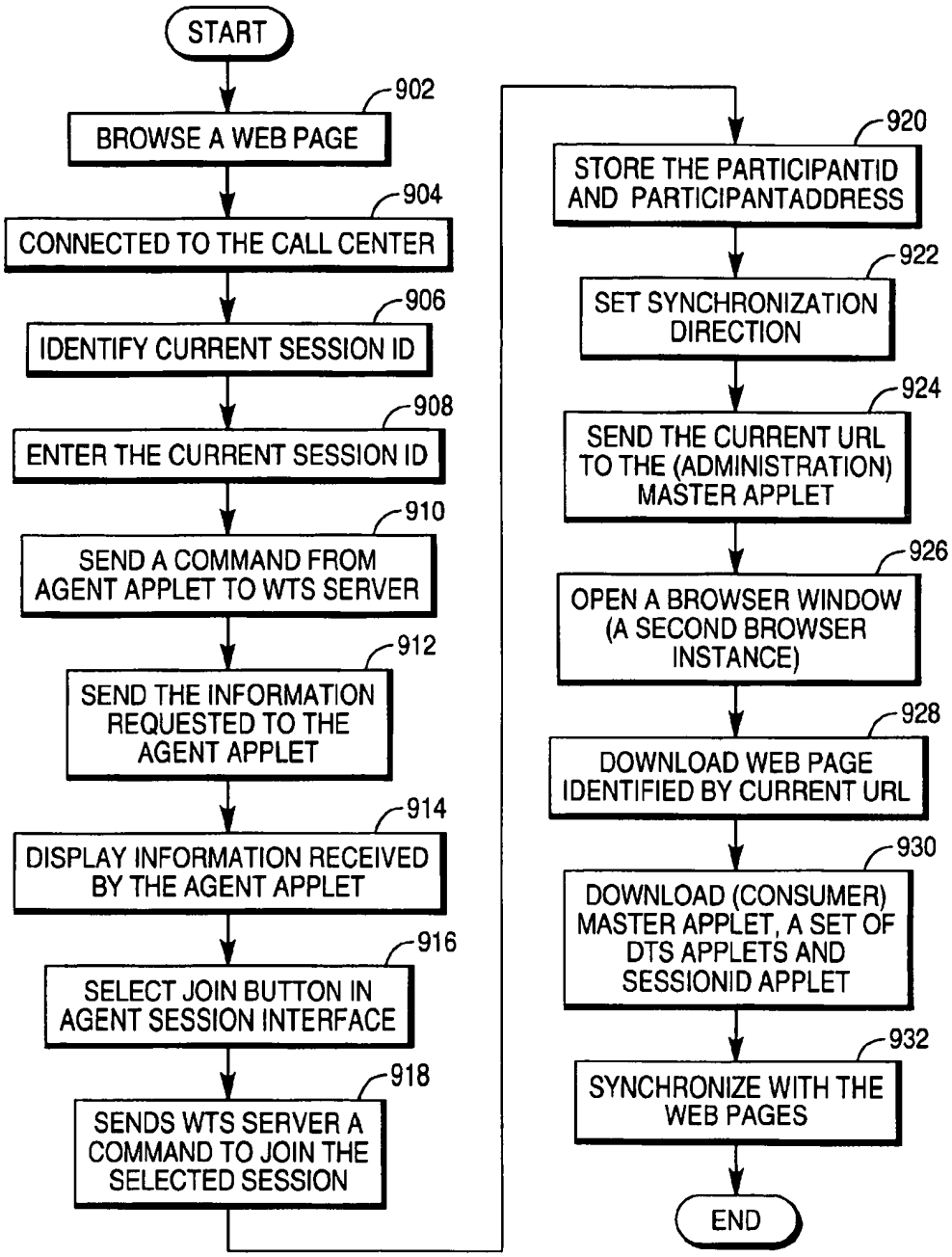

800C
858
860
Admin Supervisor Agent
Session Length
Participant count
URL Count
Current URL
URL History
Session Replay
Agent Input
Enter Session ID
Select Leadership Position
850
852
848
Leader
Follower
Drop Session
Join Session
ParticipantID
Participant Name
AgentID
844
846
856

FIG. 8C

FIG. 9


START
902
BROWSE A WEB PAGE
904
CONNECTED TO THE CALL CENTER
906
IDENTIFY CURRENT SESSION ID
908
ENTER THE CURRENT SESSION ID
910
SEND A COMMAND FROM AGENT APPLET TO WTS SERVER
912
SEND THE INFORMATION REQUESTED TO THE AGENT APPLET
914
DISPLAY INFORMATION RECEIVED BY THE AGENT APPLET
916
SELECT JOIN BUTTON IN AGENT SESSION INTERFACE
918
SENDS WTS SERVER A COMMAND TO JOIN THE SELECTED SESSION
920
STORE THE PARTICIPANTID AND PARTICIPANTADDRESS
922
SET SYNCHRONIZATION DIRECTION
924
SEND THE CURRENT URL TO THE (ADMINISTRATION) MASTER APPLET
926
OPEN A BROWSER WINDOW (A SECOND BROWSER INSTANCE)
928
DOWNLOAD WEB PAGE IDENTIFIED BY CURRENT URL
930
DOWNLOAD (CONSUMER) MASTER APPLET, A SET OF DTS APPLETS AND SESSIONID APPLET
932
SYNCHRONIZE WITH THE WEB PAGES
END

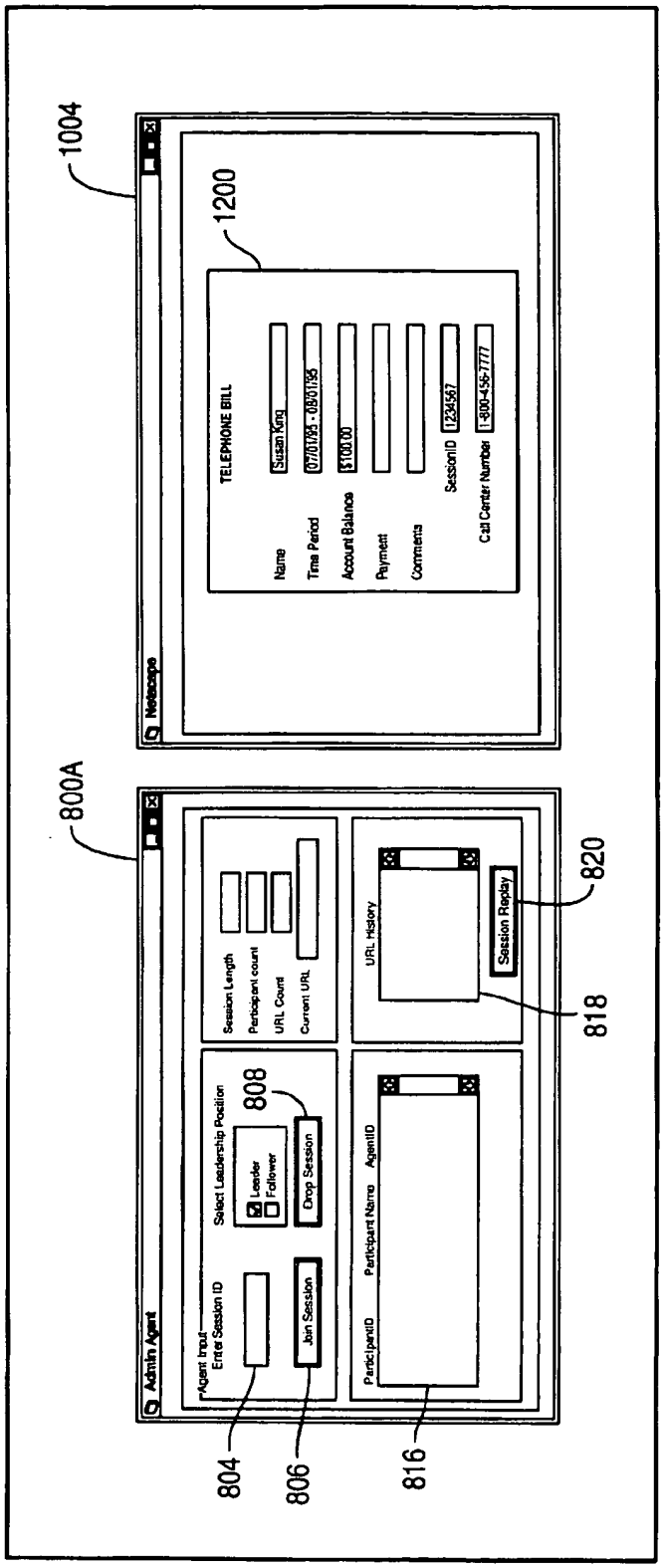

1004
1200
TELEPHONE BILL
Name
Time Period
Account Balance
Payment
Comments
SessionID
Call Center Number
Susan Kong
07/01/99 - 08/01/99
$100.00
1234567
1-800-456-7777
800A
Admin Agent
Agent Input
Enter Session ID
Select Leadership Position
Leader
Follower
Drop Session
Join Session
Session Length
Participant count
URL Count
Current URL
URL History
Session Replay
ParticipantID
Participant Name
AgentID
804
806
808
816
818
820

FIG. 10

FIG. 11

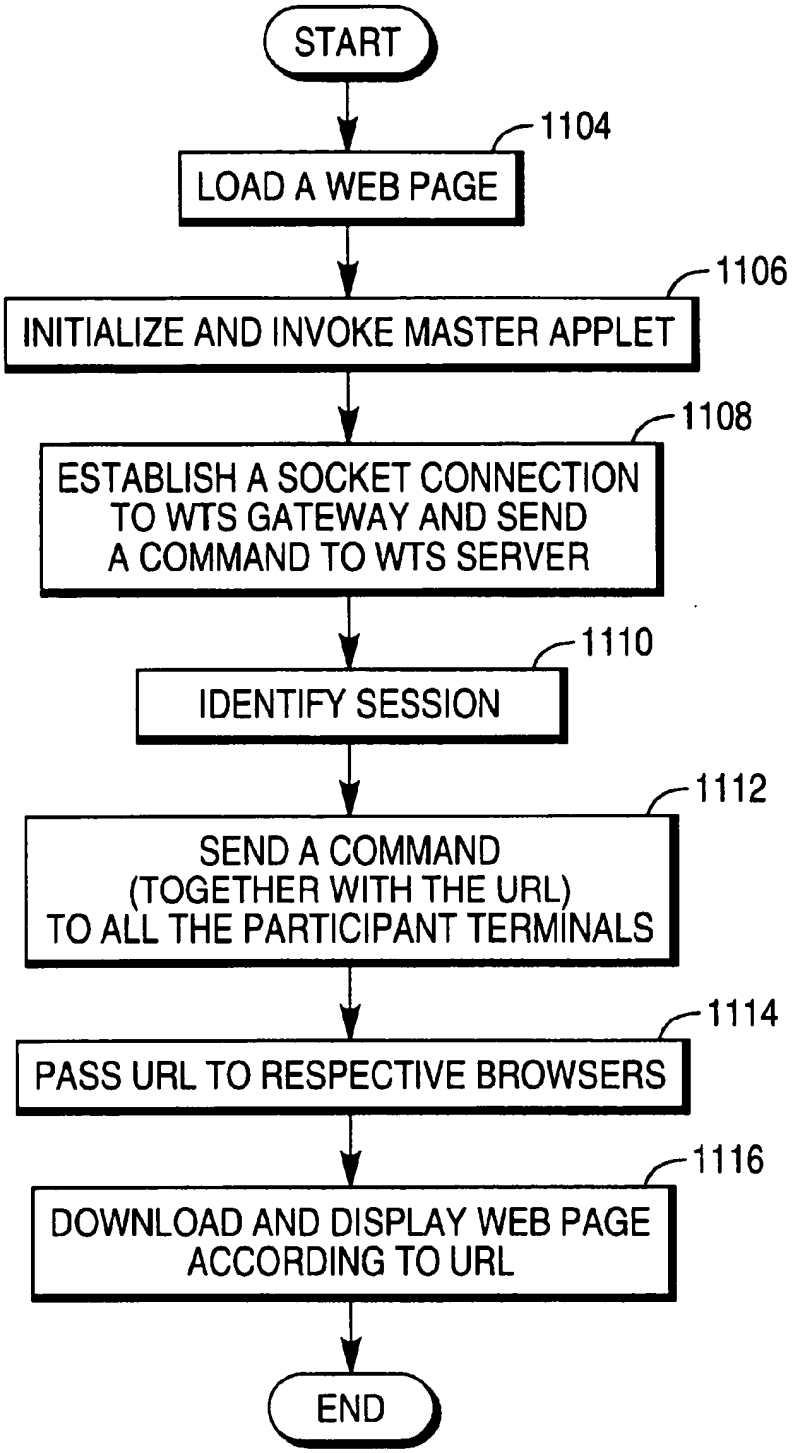

START
1104
LOAD A WEB PAGE
1106
INITIALIZE AND INVOKE MASTER APPLET
1108
ESTABLISH A SOCKET CONNECTION TO WTS GATEWAY AND SEND A COMMAND TO WTS SERVER
1110
IDENTIFY SESSION
1112
SEND A COMMAND (TOGETHER WITH THE URL) TO ALL THE PARTICIPANT TERMINALS
1114
PASS URL TO RESPECTIVE BROWSERS
1116
DOWNLOAD AND DISPLAY WEB PAGE ACCORDING TO URL
END

FIG. 12A

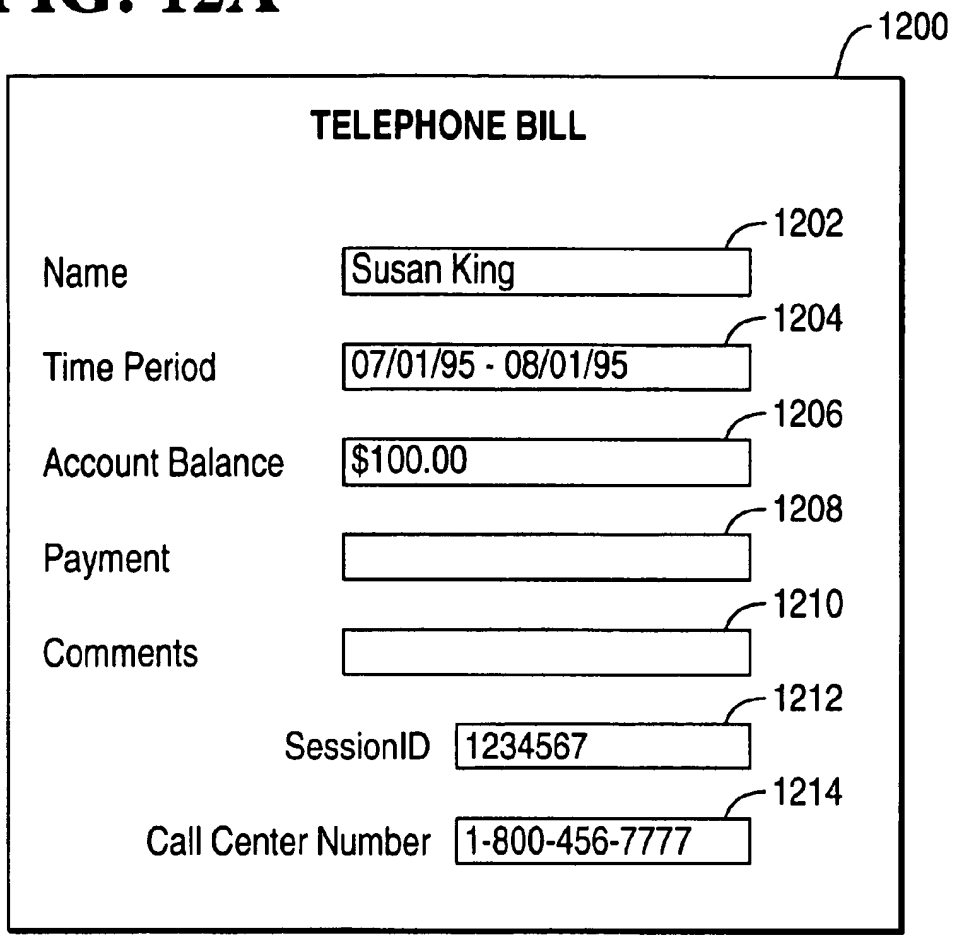

1200
TELEPHONE BILL
1202
Name
Susan King
1204
Time Period
07/01/95 - 08/01/95
1206
Account Balance
$100.00
1208
Payment
1210
Comments
1212
SessionID
1234567
1214
Call Center Number
1-800-456-7777

FIG. 12B

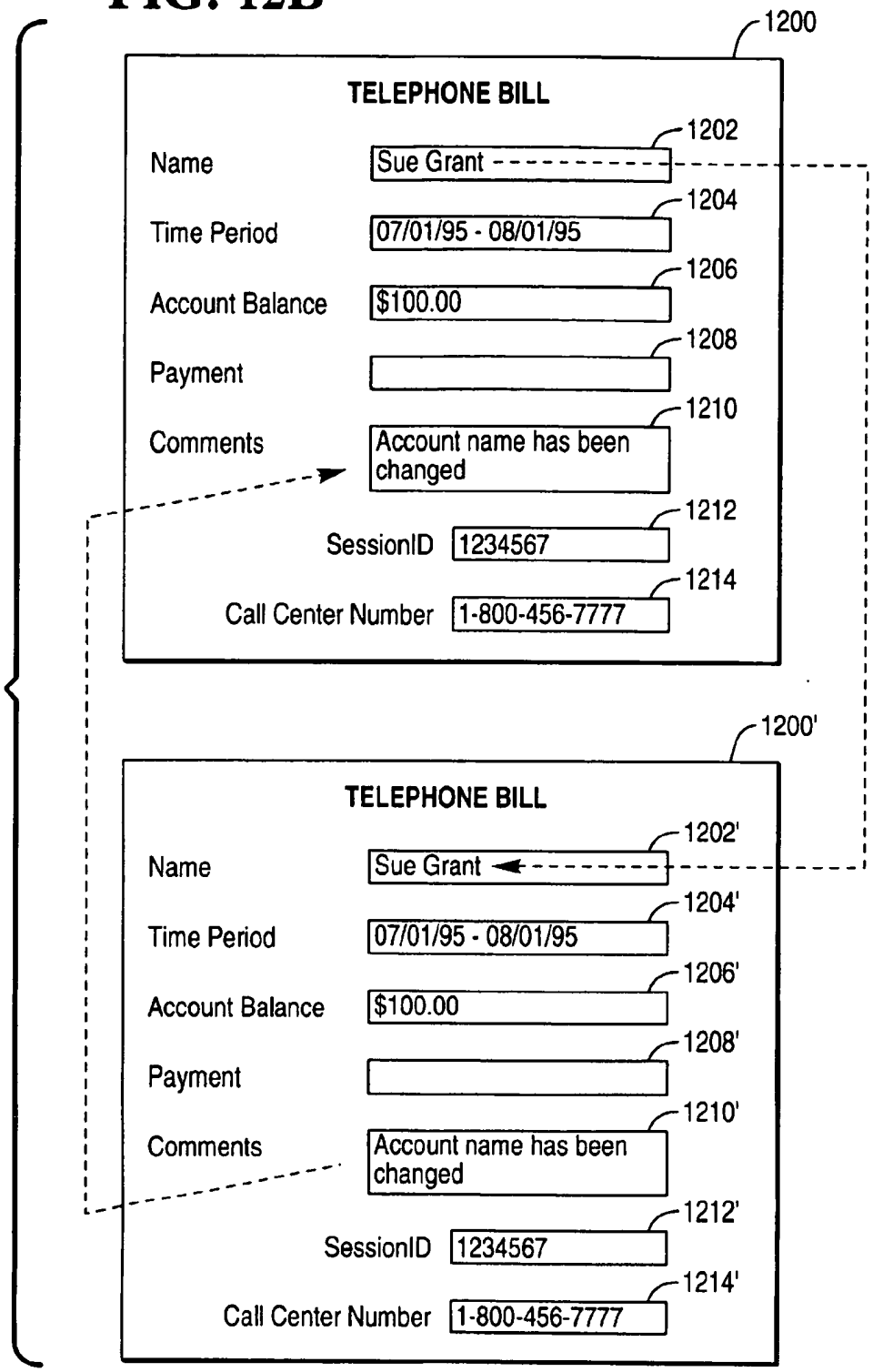

1200
TELEPHONE BILL
1202
Name
Sue Grant
1204
Time Period
07/01/95 - 08/01/95
1206
Account Balance
$100.00
1208
Payment
1210
Comments
Account name has been changed
1212
SessionID
1234567
1214
Call Center Number
1-800-456-7777
1200'
TELEPHONE BILL
1202'
Name
Sue Grant
1204'
Time Period
07/01/95 - 08/01/95
1206'
Account Balance
$100.00
1208'
Payment
1210'
Comments
Account name has been changed
1212'
SessionID
1234567
1214'
Call Center Number
1-800-456-7777

FIG. 13

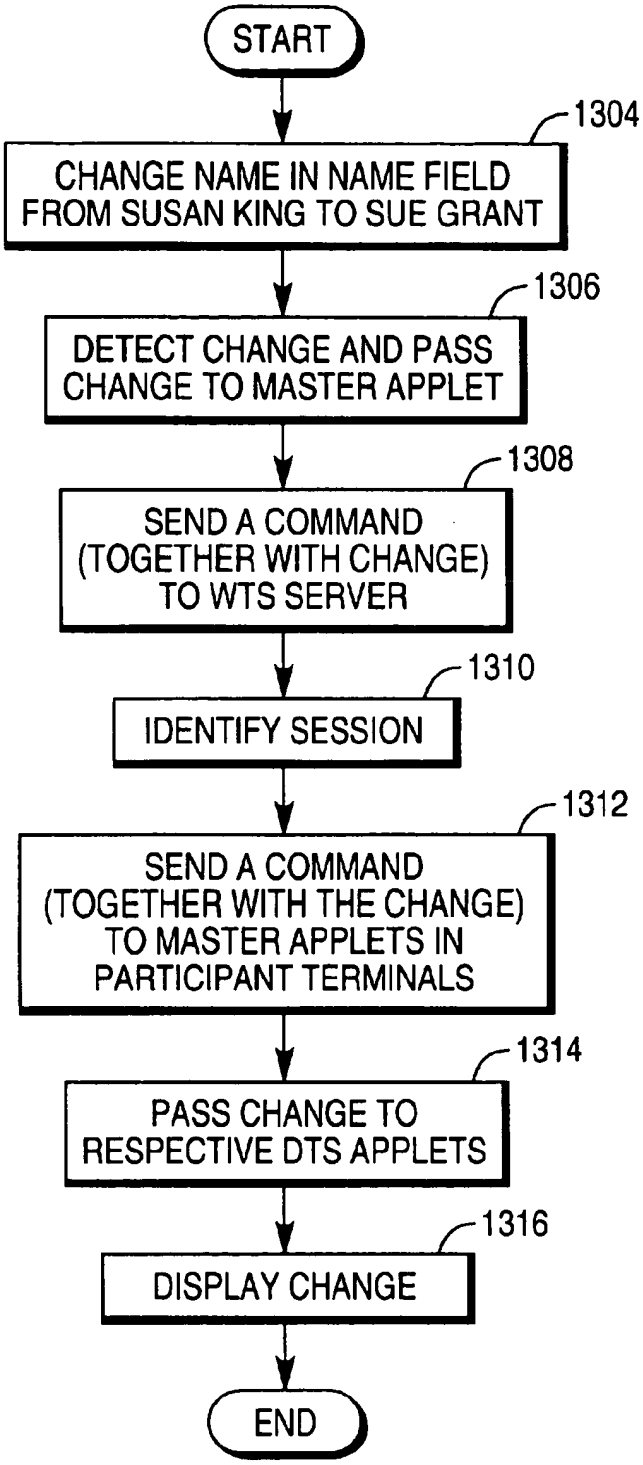

START
1304
CHANGE NAME IN NAME FIELD FROM SUSAN KING TO SUE GRANT
1306
DETECT CHANGE AND PASS CHANGE TO MASTER APPLET
1308
SEND A COMMAND (TOGETHER WITH CHANGE) TO WTS SERVER
1310
IDENTIFY SESSION
1312
SEND A COMMAND (TOGETHER WITH THE CHANGE) TO MASTER APPLETS IN PARTICIPANT TERMINALS
1314
PASS CHANGE TO RESPECTIVE DTS APPLETS
1316
DISPLAY CHANGE
END

FIG. 14

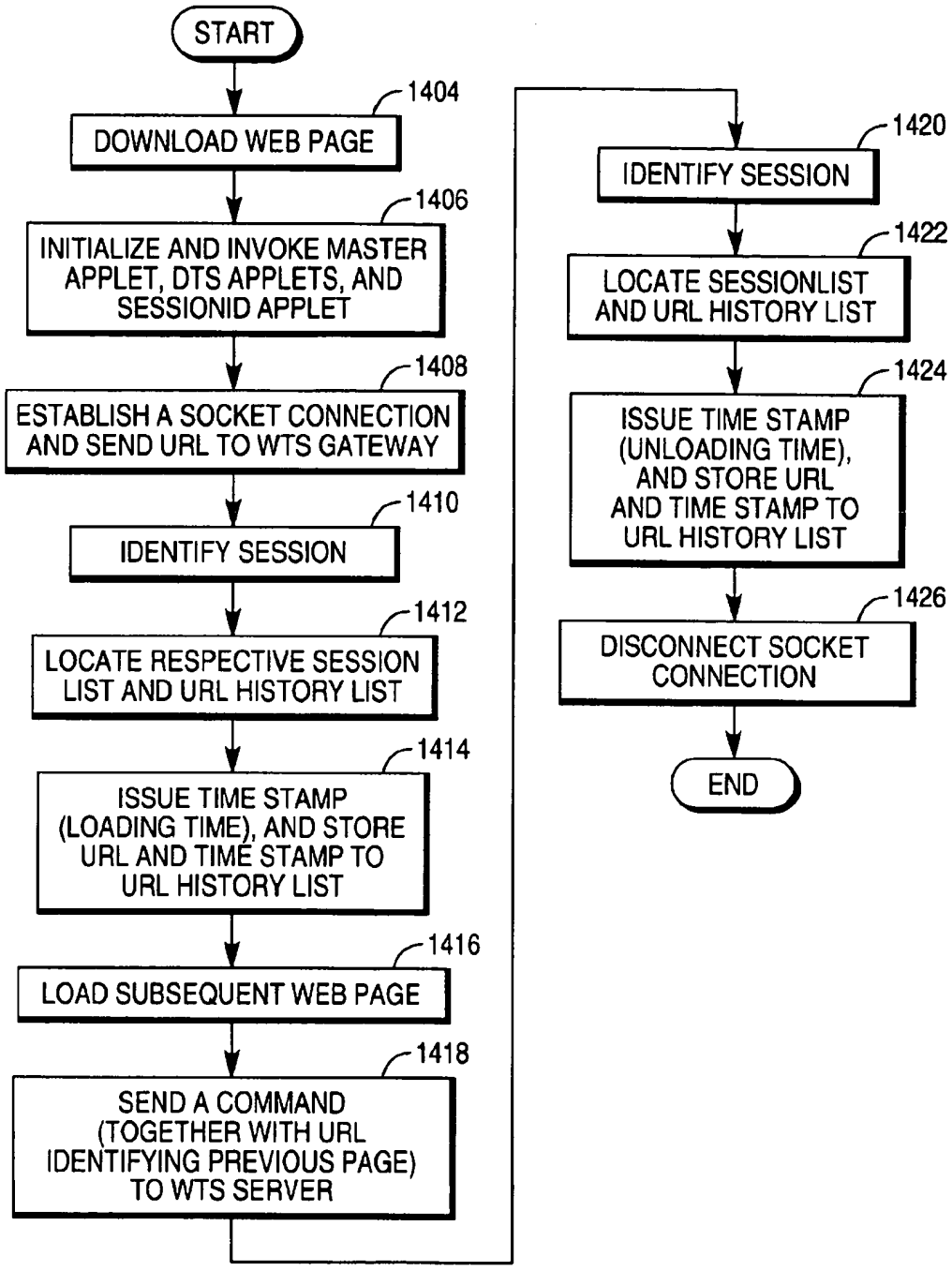

START
1404
DOWNLOAD WEB PAGE
1406
INITIALIZE AND INVOKE MASTER APPLET, DTS APPLETS, AND SESSIONID APPLET
1408
ESTABLISH A SOCKET CONNECTION AND SEND URL TO WTS GATEWAY
1410
IDENTIFY SESSION
1412
LOCATE RESPECTIVE SESSION LIST AND URL HISTORY LIST
1414
ISSUE TIME STAMP (LOADING TIME), AND STORE URL AND TIME STAMP TO URL HISTORY LIST
1416
LOAD SUBSEQUENT WEB PAGE
1418
SEND A COMMAND (TOGETHER WITH URL IDENTIFYING PREVIOUS PAGE) TO WTS SERVER
1420
IDENTIFY SESSION
1422
LOCATE SESSIONLIST AND URL HISTORY LIST
1424
ISSUE TIME STAMP (UNLOADING TIME), AND STORE URL AND TIME STAMP TO URL HISTORY LIST
1426
DISCONNECT SOCKET CONNECTION
END

FIG 15

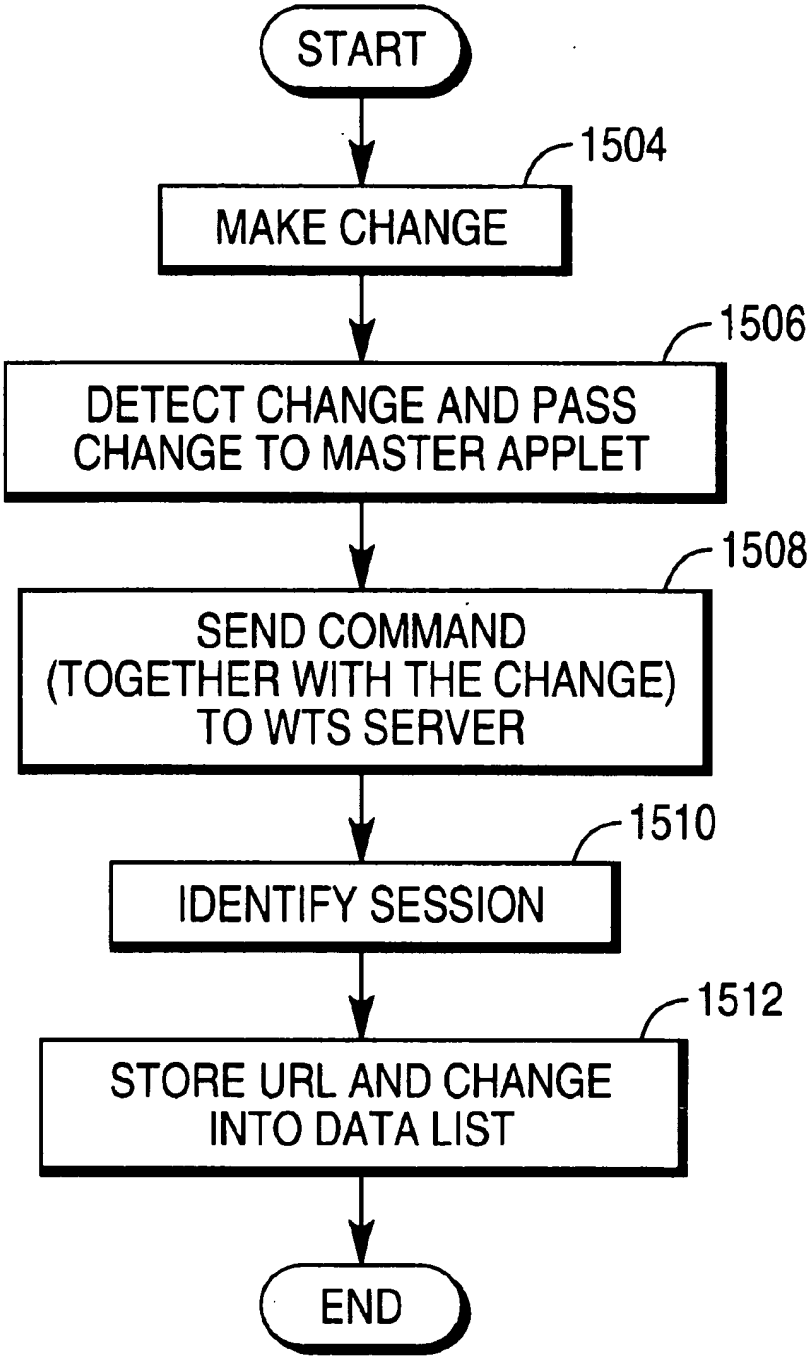

START
1504
MAKE CHANGE
1506
DETECT CHANGE AND PASS CHANGE TO MASTER APPLET
1508
SEND COMMAND (TOGETHER WITH THE CHANGE) TO WTS SERVER
1510
IDENTIFY SESSION
1512
STORE URL AND CHANGE INTO DATA LIST
END

FIG. 16

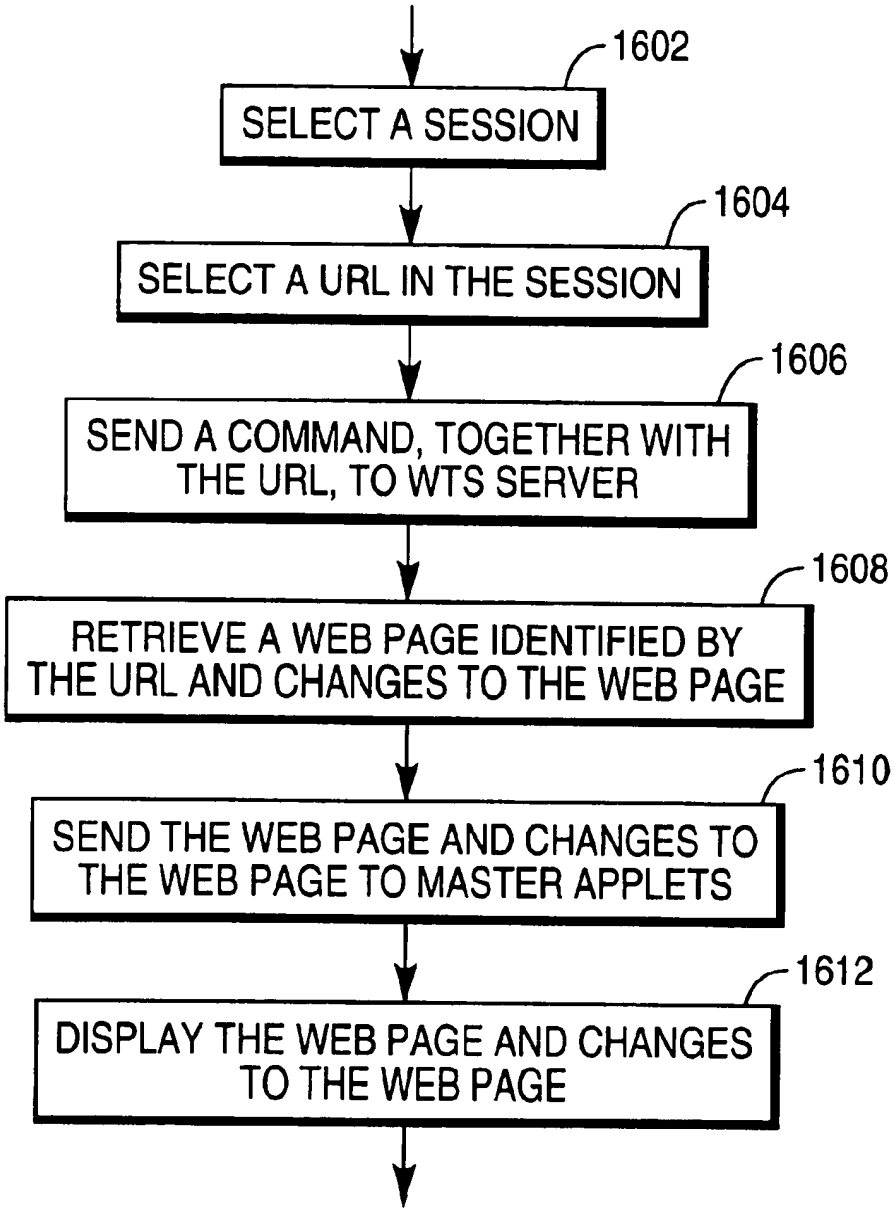
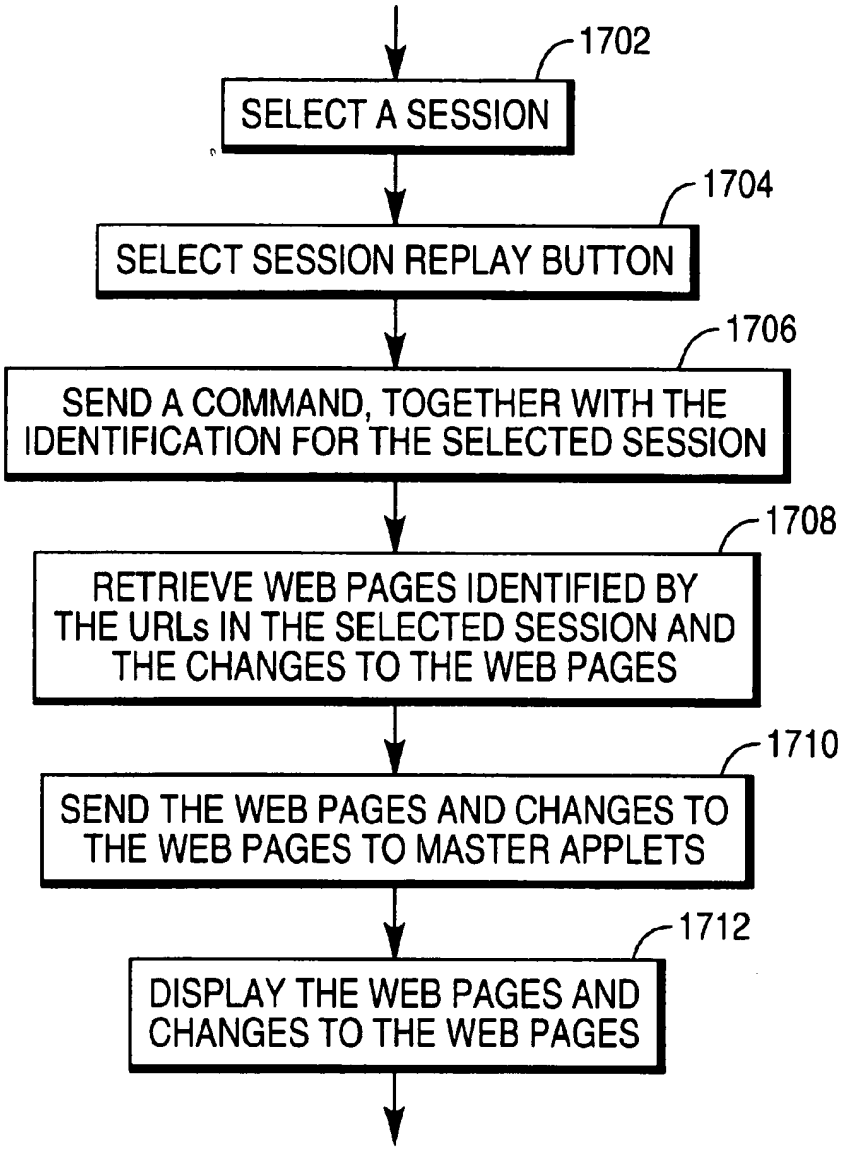

1602
SELECT A SESSION
1604
SELECT A URL IN THE SESSION
1606
SEND A COMMAND, TOGETHER WITH THE URL, TO WTS SERVER
1608
RETRIEVE A WEB PAGE IDENTIFIED BY THE URL AND CHANGES TO THE WEB PAGE
1610
SEND THE WEB PAGE AND CHANGES TO THE WEB PAGE TO MASTER APPLETS
1612
DISPLAY THE WEB PAGE AND CHANGES TO THE WEB PAGE

FIG. 17


1702
SELECT A SESSION
1704
SELECT SESSION REPLAY BUTTON
1706
SEND A COMMAND, TOGETHER WITH THE IDENTIFICATION FOR THE SELECTED SESSION
1708
RETRIEVE WEB PAGES IDENTIFIED BY THE URLs IN THE SELECTED SESSION AND THE CHANGES TO THE WEB PAGES
1710
SEND THE WEB PAGES AND CHANGES TO THE WEB PAGES TO MASTER APPLETS
1712
DISPLAY THE WEB PAGES AND CHANGES TO THE WEB PAGES

METHOD FOR RECORDING AND REPRODUCING THE BROWSING ACTIVITIES OF AN INDIVIDUAL WEB BROWSER

RELATED APPLICATIONS

This is a continuation-in-part of application Ser. No. 08/944,124 filed Oct. 6, 1997, now U.S. Pat. No. 5,951,643.

The present invention relates generally to a method and system for recording and reproducing the activities performed by an individual web browser.

BACKGROUND OF THE INVENTION

Web browsers, such as Microsoft Corporation's Internet Explorer internet browser and Netscape Communications Corporation's Navigator™ internet browser, provide a graphical, easy-to-navigate interface for retrieving and viewing information available from the Web. The World Wide Web utilizes a system known as a hypertext system to facilitate navigation through the World Wide Web environment. The hypertext system employs special connections, or links, which are embedded into documents displayed through use of internet browser software. Clicking on a word, phrase, image or thumbnail graphic including one of these links instructs the browser to retrieve a document, graphic, sound, or other information associated with the embedded link.

Documents published electronically in hypertext form, i.e., documents constructed utilizing Hypertext Markup Language (HTML), are becoming a defacto standard for sharing information online. HTML is a language for describing structured documents. Documents written in HTML are ASCII text documents including: (1) the text of the document, and (2) HTML tags that identify document elements, formatting and structure, as well as links to graphic files, other documents, or other information. Browser software interprets the HTML tags included in downloaded documents to properly display the text and referenced elements. Images and other referenced elements are not normally contained in the downloaded document, however, and must be separately downloaded in order to completely display the HTML document and referenced elements.

Documents, also referred to as web pages, as well as the image files, additional document files and other elements referenced in the web pages may be maintained on one or multiple web servers remote from the user retrieving and viewing the web pages. With the development of information technology and networking infrastructure, more and more businesses, government agencies, universities and individuals maintain web sites and conduct business over the Internet. For example, a service provider can display its service in a set of web pages maintained on a web server owned by the provider.

To improve the service and assistance provided to consumers using the Internet, it is desirable to provide a mechanism to record the detailed browsing activities of an individual browser, or to force a browser to reproduce the browser activities recorded at one or more browsers. The ability to record and reproduce browser activities is useful in a variety of applications such as consumer behavior analysis, quality assurance, and training applications.

One difficulty in achieving the above objectives is that web servers are designed to serve each individual web browsers in a stateless manner where one browser request has no relationship to previous or subsequent requests. In processing of requests, a web site has no control over the sequences of the requests.

Currently it is very difficult to accurately capture the browsing activities of a person visiting a web site. Analyzing page requests received by a web server does not accurately reflect the actual sequence of pages visited by an individual browser. This is because:

- Web pages retrieved by a browser from browser cache or a proxy server will not be recorded as a page request on the web server.
- A web server does not record the timing characteristics of a user's interaction with a form because this interaction is handled directly by the web browser. The final state of the data entered into a form is sent by the browser to the web server when the form is submitted.

Because of the limited quality (granularity) of data maintained by the server, it is currently not possible for a server-based application to reproduce the sequence of activities performed by a specific browser. Web servers currently record statistics on each page requested, but do not reliably keep track of each page that is displayed at a user's browser, or record page requests retrieved from the browser's cache or a proxy server. Nor does the web server currently record browser activities including the timing of each data entry into an HTML form. There are no known tools that are capable of recording detailed browser activities and then later providing a mechanism to replay the same sequence of browser activities through one or more browsers.

OBJECTS OF THE INVENTION

It is therefore an object of the present invention to provide a new and useful mechanism to record the detailed browsing activities of an individual browser—regardless of whether the information displayed by the browser was obtained from browser cache, proxy server cache or a web server.

It is another object of the present invention to provide such a mechanism which includes the ability to store each URL request and each piece of data entered into an HTML form along with a time stamp to keep track of when each of the browser activities took place.

It is a further object of the present invention to provide a new and useful mechanism to force a browser to reproduce a recorded set of browser activities at one or more browsers.

It is yet another object of the present invention to provide such a mechanism wherein the set of recorded browser activities can be reproduced at the same rate as the original browser activities, or can be reproduced at proportionately slower or faster rates.

It is an additional object of the present invention to provide such a mechanism wherein the playing of a sequence of web pages may be started, stopped or paused at any URL in the sequence of pages recorded.

SUMMARY OF THE INVENTION

In one aspect, the present invention provides a method for repeating browsing activities performed by a customer or user. The method comprises the steps of: (a) recording browsing activities as they are performed at a first terminal; (b) retrieving to a second terminal the browsing activities recorded at the first terminal; and (c) repeating at the second terminal the browsing activities recorded at the first terminal.

The present invention further provides a method for recording the activities of a number of customers using a

browser and then selecting one browser session for replay at another browser. The method comprises the steps of: (a) browsing web pages at a plurality of user browsers; (b) creating a plurality of sessions at a network server, each of said sessions being associated with a respective one of said plurality of user browsers; (c) recording the browsing activities for each of the user browsers into an associated session at said network server; (d) selecting one of the sessions; and (e) repeating activities recorded for the selected session at an administrative browser. We could also replay to more than one browser at the same time.

BRIEF DESCRIPTION OF THE DRAWINGS

The purpose and advantages of the present invention will be apparent to those skilled in the art from the following detailed description in conjunction with the appended drawing, in which:

FIG. **1** shows a system includes N terminals, a network, and a web site, in accordance with the present invention.

FIG. **2** shows a situation where each of the N terminals has downloaded its respective Master Applets, DTS Applets, and SessionID Applet, in accordance with the present invention.

FIG. **3** shows the process of the (consumer) Master Applet, DTS Applets, and SessionID Applet being downloaded into a terminal, in accordance with the present invention.

FIG. **4** shows the process of the (consumer) Master Applet, DTS Applets, and SessionID Applet being invoked, in response to loading a subsequent web page, to perform the operations in accordance with the present invention, when these Applets have been previously downloaded and cached in a terminal.

FIG. **5** shows the process of the (consumer) Master Applet, DTS Applets, and SessionID Applet being invoked, in response to loading a subsequent web page, to perform the operations in accordance with the present invention, when both these Applets and the web page have been previously downloaded and cached in a terminal.

FIG. **6** shows a session table in greater detail, in accordance with the present invention.

FIG. **7** shows how an agent (or supervisor) can create a session interface by downloading an agent page (or a supervisor page) from administration page repository **149**, in accordance with the present invention.

FIG. **8**A shows an agent session interface, in accordance with the present invention.

FIG. **8**B shows a browser supervisor session interface, in accordance with the current invention.

FIG. **8**C shows a supervisor agent session interface, in accordance with the present invention.

FIG. **9** shows a flowchart illustrating the operation of joining a session by an agent, in accordance with the present invention.

FIG. **10** shows a screen display containing two browse instances, in accordance with the present invention.

FIG. **11** shows a flowchart illustrating the operation of web page synchronization, in accordance with the present invention.

FIG. **12**A shows a web page containing five data fields, in accordance with the present invention.

FIG. **12**B shows a web page that is similar to that of FIG. **12**A, except that the data in one of the five data fields is changed, in accordance with the present invention.

FIG. **13** shows a flowchart illustrating the operation of data synchronization, in accordance with the present invention.

FIG. **14** shows a flowchart illustrating the operation of web page tracking, in accordance with the present invention.

FIG. **15** shows a flowchart illustrating the operation of data tracking, in accordance with the present invention.

FIG. **16** shows a flowchart illustrating the operation of re-browsing a web page previously browsed in a session, in accordance with the present invention.

FIG. **17** shows a flowchart illustrating the operation of re-browsing all web pages previously browsed in a session, in accordance with the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the preferred embodiments will be readily apparent to those skilled in the art, and the principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded with the broadest scope consistent with the principles and features disclosed herein.

Referring to FIG. **1**, there is shown an exemplary web page tracking and synching system **100**, in accordance with the present invention.

As shown in FIG. 1, the system includes N terminals **104A, 104B, . . . ,104N**; a network **129** such as the Internet, or a combination of the Internet and an Intranet and a web site **134**. Each of the terminals has a telephone set or an alternate means of communications such as an ip phone, chat application, etc., **102A, 102B, . . . , 102N** installed in its vicinity. Each of the terminals can be a PC computer, a workstation, a Java station, or even a web TV system.

Web site **134** includes a WTS (Web Tracking and Synching) gateway **142**, a WTS server **144** containing a session table **145** and a user class table **147**, a database processing application **148**, an HTTP (Hyper Text Transfer Protocol) server **152**, and a hard disk unit **154** for storing consumer page repository **146**, administration page repository **149**, and database **156**. All the components in web site **134** can be installed in one or distributed across one or more computer systems. Each of the computer systems includes a processing unit (which may include a plurality of processors), a memory device, and a disk unit (which may include a plurality of disk sets).

Each of the terminals **104A, 104B, . . . ,104N**, includes a processor unit (not shown) and a memory area **115A, 115B, . . . , 115N**, and runs a Java enabled web browser **114A, 114B, . . . ,114N**. Each of the memory area **115A, 115B, . . . , 115N**, is maintained by its respective browser **114A, 114B, . . . , 114N**. Via network **129**, each of the browsers **114A, 114B, . . . ,114N**, is able to send requests to and receive web pages from HTTP server **152**, and to display the web pages received at its respective terminal. Each of the browsers **114A, 114B, . . . ,114N**, is able to run a Master Applet **124A, 124B, . . . ,124N**, a set of DTS (Data Tracking and Synching) Applets, a SessionID Applet, and an Agent Applet. As shown in FIG. 1, these Applets are stored in consumer page repository **146** and can be downloaded from consumer page repository **146** and stored in the memory

areas (otherwise known as browser cache) of the terminals **104A, 104B, . . . , 104N.**

Referring to FIG. **2**, there is shown the situation where each of the terminals **104A, 104B, . . . ,104N**, has downloaded its respective Master Applets **124A, 124B, . . . ,124N**, DTS Applets **126A, 126B, . . . , 128N**, and SessionID Applets **128A, 128B, . . . , 128N**, in accordance with the present invention.

In FIG. **2**, each of the (consumer) Master Applets **124A, 124B, . . . ,124N**, is primarily responsible for: (1)establishing a socket connection to WTS gateway **142** via network **129** for its respective browser **114A, 114B, . . . ,114N**, in response to loading each web page at its respective browser, (2) communicating with WTS server **144** via the socket connection, from which WTS server **144** is able to identify the origin, i.e., which browser, which web page, etc., of the commands and information that are being delivered through, (3) monitoring the activities of its respective browser, (4) sending the information about its respective browser's activities to WTS server **144**, (5) receiving and processing the information about other browsers' activities, (6) via the socket connection, providing a single communication path to WTS server **144** for DTS Applets **126A, 126B, . . . , 126N**, SessionID Applets **128A, 128B, . . . , 128N**, or any other consumer Applets embedded on the same page with the Master Applet, (7) sending commands to WTS server **144** to request services, for itself and for DTS Applets **126A, 126B, . . . , 126N**, SessionID Applets **128A, 128B, . . . , 128N**) , or any other consumer Applets embedded on the same page with the Master Applet, and (8) sending user class information together with the commands, to indicate that its respective browser is a consumer user.

Each set of DTS Applets **126A, 126B, . . . , 126N**, contains one or more individual DTS Applets, which are primarily responsible for: (1) displaying and monitoring the data activities (data inputs or data updates of data fields) on web pages that are being displayed by its respective browser, (2) sending the data activities to WTS server **144** via its respective Master Applet, (3) receiving the data activities from other browsers via its respective Master Applet, and (4) processing the data activities from other browsers for the web pages that are being displayed by its respective browser.

Each of the SessionID Applets **128A, 128B, . . . , 128N**, is responsible for retrieving, and for displaying on a web page the current SessionID.

As shown in administration page repository **149**, Agent Applet (or Supervisor Applet) is responsible for creating a session interface, joining, monitoring, and controlling a session through the session interface. The (administration) Master Applet is primarily responsible for: (1) opening a dedicated socket, and establishing a socket connection to WTS gateway **142** via network **129** for the session interface created by Agent Applet, Supervisor Applet, or any other administration Applets embedded on the same web page with the Master Applet, (2) communicating with WTS server **144** via the socket connection, from which WTS server **144** is able to identify the origin (i.e. from which session interface) of the commands and information that are being delivered through, (3) via the socket connection, providing a single communication path to WTS server **144** for Agent Applet, Supervisor Applet, or any other administration Applets embedded on the same web page with the Master Applet, and (4) sending user class information together with the commands, to indicate that its respective browser is an administration user.

WTS gateway **142** is responsible for maintaining all socket connections between Master Applets and WTS server **144**. The connections between Master Applets and WTS gateway **142** take place using standard sockets. In the implementation described herein, the connection between WTS gateway **142** and WTS server **144** takes place using Java RMI (Remote Method Invocation). This connection could also be accomplished utilizing sockets.

WTS server **144** is responsible for: (1) managing and tracking the activities of all browsers participating in active sessions, exemplary activities including: loading of, interacting with, and unloading of web pages, (2) recording the information about the activities, (3) managing the synchronization of the activities for all browsers participating in the active sessions, (4) creating a session when a consumer user (via a browser) sends a request to web site **134** for the first time, (5) defining the session length intervals, (6) purging sessions that have been inactive for more than the specified session length intervals, (7) adding and deleting participants to a session, and (8) providing services to all commands from consumer Applets, such as: (consumer) Master Applet, DTS Applets, SessionID Applets, and administration Applets, such as: (administration) Master Applet, Agent Applets, and Supervisor Applet.

Consumer page repository **146** stores the web pages and Applets for consumers. Consumer Applets can be selectively embedded into consumer web pages. Exemplary consumer Applets include: (consumer) Master Applet, DTS Applets, SessionID Applet, etc.

Administration page repository **149** stores the web pages and Applets for call center administration users, including: administrator, supervisor, agent, etc. Administration Applets can be selectively embedded into administration web pages. Exemplary administration Applets include: (administration) Master Applet, Agent Applet, Supervisor Applet, etc.

To better describe the present invention, the Applets stored in (or downloaded from) repository **146** can be referred to as consumer Applets, and the Applets stored in (or downloaded from) repository **149** can be referred to as administration Applets. HTTP server **152** contains a security application that allows consumer users to get access only to the web pages stored in consumer page repository **146**, and allows administration users (such as administrator, supervisor, agent, etc.) to get access to the web pages stored in both consumer page repository **146** and administration page repository **149**. Note that because an agent can access consumer applets, there is no need to have separate master applets stored in administration and consumer repositories. A single master applet could be utilized to perform both purposes.

Session table **145** is responsible for maintaining the information for all active sessions.

Class table **147** is responsible for keeping records of user classes or types assigned to different users. Listed are exemplary user types: administrator, supervisor, agent, and consumer.

Based on user classes (administrator, supervisor, agent, and consumer), WTS server **144** provides the following services:

(1) creating a session (consumer);
(2) storing data received from a session participant (supervisor, agent, and consumer);
(3) listing active sessions (administrator and supervisor);
(4) listing the information associated with active sessions (administrator, and supervisor);
(5) listing current users (administrator);
(6) joining a session (supervisor and agent);
(7) terminating a session (supervisor);

(8) monitoring a session (supervisor and agent);
(9) configuring a session parameters (administrator); and
(10) sending commands and information to a consumer Master Applet or an administration Master Applet in a participating browser (supervisor, agent, and consumer).

Database **156** is responsible for storing data collected in session table **145**.

HTTP server **152** is responsible for processing the requests issued by one of the web browsers, retrieving the web pages from either consumer page repository **146** or administration page repository **149**, and sending the web pages to the browsers that have generated these requests.

Database processing application **148** is responsible for writing the data collected in session table **145** into database **156**.

Referring to FIG. **3**, there is shown the process of how the (consumer) Master Applet, DTS Applets, and SessionID Applet being downloaded into terminal **104A** from HTTP server **152** in response to loading an initial web page, and then being invoked to perform the operations in accordance with the present invention.

As shown in FIG. **3**, a (consumer) Master Applet, a set of DTS Applets, and a SessionID Applet are embedded into web page **204** by using a set of applet tags **208**. Web page **204** is associated with a specific URL indicating the location of web page **204** in HTTP server **152**.

As indicated by dotted line (1), web browser **114A** sends a request including the URL of web page **204** to HTTP server **152** via network **129** (shown in FIG. **2**). As indicated by dotted line (2), in response to the request, HTTP server **152** retrieves web page **204** from consumer page repository **146** and sends it to web browser **114A** via network **129**. Web page **204** contains a set of applet tags **208**, which indicate the location of Master Applet, DTS Applets, and SessionID Applet in HTTP server **152**. As indicated by dotted line (3), web browser **114A** loads web page **204**. As indicated by dotted line (4), since Master Applet, DTS Applets, and SessionID Applet have not been downloaded, web browser **114A** sends requests via network **129**, to download these Applets based on applet tags **208**. As indicated by dotted line (5), HTTP server **152** sends Master Applet, DTS Applets, and SessionID Applet to browser **114A** via network **129**. As indicated by dotted line (6), browser **114A** stores Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** into memory area **115A** of terminal **104A**, and initializes and invokes these Applets. After being invoked, these Applets are running together with web browser **114A**, to monitor and process the activities for which they are assigned to be responsible. As indicated by line (7), Master Applet **124A** opens a dedicated socket and establishes a socket connection to WTS gateway **142** for browser **114A** and web page **204**. Via the socket connection, Master Applet **126** sends WTS server **144** a command, together with an ID unique to browser 1 **14A**. In response to the command from Master Applet **126**, WTS server **144** creates a session for browser **114A** based on the unique ID, and issues a time stamp (loading time) indicating the time at which the command was received, and stores the URL and time stamp of web page **204** into the session created for browser **114**. As will be seen in the description in connection with FIG. **6** following, the URL, command, and loading time are stored in a URL history list and a command list created for the session.

Referring to FIG. **4**, there is shown the process of the (consumer) Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** being invoked, in response to loading a subsequent web page **214** (subsequent to web page **204**), to perform the operations in accordance with the present invention, when these Applets have been previously downloaded and cached in terminal **104A**.

As indicated by dotted line (1), to download web page **214**, web browser **114A** sends a request including the URL of web page **214** to HTTP server **152** via network **129**. Before loading web page **214**, the following events occur: (a) browser **114A** instructs Master Applet **124A** to run a stop routine, (b) via the socket connection established for browser **114A** and web page **204**, Master Applet **124** sends a command to inform WTS server **144** that web page **204** has been unloaded, and disconnects the socket connection established for browser **114A** and web page **204**, (c) WTS server **144** issues a time stamp (unloading time) indicating the time the command was received, and (d) records the URL and the time stamp of web page **204** into the session created for browser **114A**. As shown in FIG. **6** and described in greater detail below, the URL, command, and unloading time are stored in a URL history list and a command list created for the session. As indicated by dotted line (2), HTTP server **152** retrieves web page **214** from consumer page repository **146** and sends it to browser **114A**. Like web page **204**, web page **214** contains a set of applet tags **208** for indicating the location of Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A**. As indicated by dotted line (3), web browser **114A** loads web page **214**. As indicated by dotted line (4), in response to the loading of web page **214**, web browser **114A** locates Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** (based on the indication of applet tags **208**) that are cached by browser **114A** in memory area **115A**, and loads these Applets and then invokes them. As indicated by line (5), Master Applet **124A** opens a dedicated socket and establishes a socket connection to WTS gateway **142** for browser **114A** and web page **214**. Via the socket connection established for browser **114A** and web page **214**, Master Applet **126A** sends a command, together with the ID unique to browser **114A** and the URL of web page **214**, to inform WTS server **144** that web page **214** has been loaded. WTS server **144** issues a time stamp (loading time) indicating the time the command was received and stores the URL and time stamp of web page into the session created for browser **114A**. shown in FIG. **6** and described in greater detail below, the URL, command, and loading time are stored in a URL history list and a command list created for the session.

Referring to FIG. **5**, there is shown the process of the (consumer) Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** being invoked, in response to loading web page **204**, to perform the operations in accordance with the present invention, when both these Applets and web page **204** have been previously downloaded and cached by browser **114A** in terminal **104A**.

As indicated by dotted line (1), web browser **114A** loads web page **204** cached in memory area **115A** maintained by browser **114A**. As described earlier, web page **224** contains a set of applet tags **208** indicating the location of Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A**. Before loading web page **224**, the following events occur: (a) browser **114A** instructs Master Applet **124A** to run a stop routine, (b) Master Applet **124A** sends a command via the socket connection established for browser **114A** and web page **214A** to inform WTS server **144** that web page **214** has been unloaded, and disconnects the socket connection established for browser **114A** and web page **214**, (c) WTS server **144** issues a time stamp (unloading time) indicating the time the command was received, and (d) WTS server **144** records the URL and time stamp of web page **214** into the session

created for browser 114A. As will be seen in the description in connection with FIG. 6, the URL, command, and unloading time are stored in a URL history list and a command list created for the session. As indicated by dotted line (2), browser 114A in response to the loading of web page 224, locates Master Applet 124A, DTS Applets 126A, SessionID Applet 128A that have been cached by browser 114A in memory area 115A in terminal 104A, and initializes and invokes these Applets. As indicated by line (3), Master Applet 124A opens a dedicated socket and establishes a socket connection to WTS gateway 142 for browser 114A and web page 224. Via the socket connection established for browser 114A and web page 224, Master Applet 126A sends a command, together with the ID unique to browser 114A and the URL of web page 224, to inform WTS server 144 that web page 224 has been loaded. WTS server 144 issues a time stamp (loading time) indicating the time the command was received and stores the URL and time stamp into the session created for browser 114. As will be seen in the description in connection with FIG. 6, the URL, command, and loading time are stored in a URL history list and a command list created for the session.

In the example shown in FIG. 5, it should be appreciated that even through no request arrives at HTTP server 144 when web page 204 is loaded from cached memory in terminal 104A, Master Applet 124A still sends browsing activities to WTS server 144.

It should be noted that the processes shown in FIGS. 3–5 of loading and invoking Master Applet, DTS Applets, and SessionID Applet for terminal 104A can also be used for terminals 104B through 104N.

In FIGS. 3–5, Master Applet, DTS Applets, and SessionID Applet are all embedded into web pages 204 and 214. However, it should be noted that there is no requirement that all of these Applets be embedded into a web page. Depending on the desired functions to be performed, respective Applets can be selectively embedded into a web page by selectively setting applet tags in the web page. For example, if data synchronization and tracking of individual elements are not needed, the applet tags for linking DTS Applets can be eliminated from the web page. By the same token, if additional functions are needed, additional applet tags can be added into the web page to link additional Applets.

Referring to FIG. 6, there is shown session table 145 (see FIG. 1) in greater detail, in accordance with the present invention.

While browsers at their respective terminals are browsing through the web pages in web site 134, WTS server 144 collects and analyzes the information about the interactions between all browsers and the web pages that have been downloaded to the browsers from web site 134. One difficulty in collecting and analyzing such information is that browsing individual web pages in web site 134 is a stateless process. More specifically, web site 134 receives a sequence of requests from different browsers, and sends the respective web pages to the respective browsers in response to the sequence requests. Since in processing the requests from an individual browser, web site 134 does maintain a constant connection to the same browser to keep an one-to-one relationship, web site 134 has no control over, and does not maintain data on, the sequences of the requests from the browsers.

To meaningfully collect and analyze the information about the interactions between the browsers and web pages, a session is defined as a collection of web page interactions that occur over a given period of time from a specific browser. A session is created when a browser first hits web site 134, and a session window (or session length interval) is defined for the session. If activities from a specific browser (identified by an ID unique to the browser, issued by a respective Master Applet) does not occur within the session window, the session is terminated and cleaned up by WTS server 144. A session window is refreshed (reset to time zero) each time the information about the associated browser is sent to WTS server 144. For example, if a session window is defined as 15 minutes, as long as the associated terminal has some activity every 15 minutes, the session will remain open. After 15 minutes of inactivity, the session is terminated. A subsequent request from the same terminal will cause a new session to be created. After a session has been created for a terminal, one or more other terminals can join the session.

As shown in FIG. 6, session table 145 includes M Session IDs created for M sessions respectively. Each of the session ID is associated with: (1) a session list for maintaining information about a session, (2) a participant list for maintaining information about all participant browsers in a session (note: when a session is first created, it only contains one participant), (3) a URL history list for maintaining information about all web pages visited by all participants in a session, (4) a data list for maintaining information about the data fields on the web pages visited by all participants in a session, and (5) a command list for maintaining information about all commands issued to WTS server 144 by the various participants in a session.

Typical items in a session list are: (1) SessionID for identifying a session, (2) UserName for indicating the actual name for whom the session is created, (3) StartTime for indicating the time of starting the session, (4) StopTime for indicating the time of stopping the session, and (5) SessionNotes for recording the notes of the session.

Typical fields contained in a participant list are: (1) SessionID for linking the participant list to a session, (2) ParticipantID for identifying a participant, (3) ParticipantAddresses for indicating a participant's IP address, (4) Class for indicating the user class of the participant (customer, agent, supervisor, administrator, etc.) and (5) Direction for indicating the synchronization direction for the participant browser.

Typical fields contained in a URL history list are: (1) SessionID for linking the URL history list to a session, (2) ParticipantID for identifying a participant who visited the web page, (3) LoadingTime for indicating the loading time of the web page, and (4) UnloadingTime for indicating the unloading time of the web page, and (5) PageURL for indicating the URL of a web page visited.

Typical fields contained in a data list are: (1) SessionID for linking the data list to a session, (2) ParticipantID for indicating the participant browser who updated this data field, (3) FieldName for indicating the actual name of the data field, (4) DataName for indicating the name of the data field displayed on a web page, (5) DataValue for indicating the value of the data field, (5) Page URL for indicating the web page on which the data field was displayed, (6) TimeStamp for indicating the time at which this data field is updated, and (7) WasRelayed for indicating if this data field has been broadcasted.

Typical fields contained in a command list are: (1) SessionID for linking the data list to a session, (2) ParticipantID for indicating the participant browser issuing the command, (3) Command for indicating the specific command executed (loading a page, unloading a page, changing a data field, etc.), (4) Page URL for indicating the web page to which the command operated, (5) FieldPoint for indicating the data

field to which the command operated, and (6) TimeStamp for indicating the time at which command was executed.

Before a session is purged from session table **145**, database processing application **147** stores the associated session list, URL history list, and command list to database **156**. The data contained in these three lists can be used by data warehouse integration applications.

Referring to FIG. **7**, there is shown an operation for creating a session interface for an agent (or a supervisor) by downloading an agent page (or a supervisor page) from administration page repository **149**, in accordance with the present invention. In the example shown in FIG. **7**, it is assumed that administration user class (either agent user class or supervisor user class) is assigned to terminal **104N**, so that the security application in HTTP server **152** grants the access to the web page stored in both consumer page repository **146** and administration page repository **149**.

At step **702**, an agent at terminal **144N** types in an agent URL at terminal **104N**, and browser **114N** sends the URL to HTTP server **152**, to retrieve an agent page, in which an (administration) Master Applet and an Agent Applet are embedded. For a supervisor, he/she types in a supervisor URL at terminal **104N**, and browser **114N** sends the URL to HTTP server **152**, to retrieve a supervisor page, in which an (administration) Master Applet and a Supervisor Applet are embedded.

At step **704**, HTTP server **152** retrieves the agent page (or a supervisor page) from administration page repository **149** and sends it to browser **114N**.

At step **706**, browser **114N** downloads the agent page, in which a Master Applet (administration Master Applet) and an Agent Applet are embedded; or downloads the supervisor page, in which a Master Applet (administration Master Applet) and a Supervisor Applet are embedded.

At step **708**, browser **114N** downloads the Master Applet and Agent Applet from HTTP server **152**, initializes and invokes these Applets; or downloads the Master Applet and Supervisor Applet from HTTP server **152**, initializes and invokes these Applets.

At step **710**, Master Applet opens a dedicated socket, establishes a socket connection to WTS gateway **142**, and sends an unique ID to WTS server **144**. WTS server **144** is able to identify browser **114N** based on the unique ID.

At step **712**, Agent Applet creates an agent session interface **800A** shown in FIG. **8A** for the agent user; or Supervisor Applet creates a supervisor session interface **800B** shown in FIG. **8B** for the supervisor agent.

Referring to FIG. **8A**, there is shown an agent session interface **800A** created for an agent at step **712**, in accordance with the present invention.

As shown in FIG. **8A**, the session interface contains a text box **804** for entering a session ID, a Join session button **806** for joining a session identified by the session ID, a drop button **808** for leaving a session, a leader check box **810** (selecting of which designates a browser as a leading browser in synchronization), a follower check box **812** (selecting of which designates a browser as a following browser in synchronization), a scrollable list box **816** for displaying the information contained in the participant list associated with a selected session, a scrollable list box **818** for displaying the information in an identified URL history list, and a Display URL Button **820** for displaying URLS selected in URL History list **818**. If both the leader and follower check boxes **810** and **812** are selected in the agent session interface, browser **114A** acts as both leading and following browser in synchronization.

Referring to FIG. **8B**, there is shown a browser supervisor session interface **800B** created for a supervisor at step **712**, in accordance with the current invention.

As shown in FIG. **8B**, the session interface contains a scrollable list box **832** for displaying session IDs of all active sessions in session table **145** and for selecting one of the session IDs, a text box **834** for displaying relevant statistics of WTS server **144**, a multi column scrollable list box **836** for displaying details about the session selected in scrollable list box **832**, a select session button **838** for selecting a session from scrollable list box **832**. By using the information in scrollable list box **832**, a supervisor agent can monitor all active sessions. By using the information in multi column scrollable list box **836**, a supervisor can monitor operational status of a session selected from scrollable list box **832**, including: (1) whether this session is being helped by an agent, (2) user name, and (3) agent ID. By selecting select session button **838**, a supervisor can access an agent session interface as shown in FIG. **8C**.

Referring to FIG. **8C**, there is shown a supervisor agent session interface **800C**, in accordance with the present invention.

Referring to FIG. **9**, there is shown a flowchart illustrating the operation of joining a session by an agent, in accordance with the present invention.

In the example shown in FIG. **9**, it is assumed that: (1) a consumer at terminal **104A** is browsing web pages from consumer page repository **146** via browser **114A**, (2) session list **1** shown in FIG. **6** has been created for browser **114A**, (3) an agent class has been assigned to browser **114N**, (4) agent session interface **800A** shown in FIG. **8A** has been displayed on terminal **104N**; (5) a (administration) Master Applet and Agent Applet have been previously downloaded into browser **114N**, (6) a dedicated socket connection has been established for session interface **800A** displayed at terminal **104N** by the (administration) Master Applet, and (7) the agent at terminal **104A** is on duty at a call center.

As shown in FIG. **9**, at step **902**, the consumer is browsing a web page at terminal **104A**. On the web page, SessionID Applet **128A** displays the current session ID. A call center telephone number the consumer can call is also displayed on the web page.

At step **904**, the consumer is connected to the call center by dialing the telephone number via telephone **102A** (see FIG. **1**), and the call is directed by the call center to the agent.

At step **906**, the consumer tells, via telephone **102A** (see FIG. **1**), the agent the current session ID displayed. It should be noted that, instead of using the telephone, the agent can be informed of the current session ID by alternative methods. For example, the consumer can enter his/her telephone number into a special web page that contains a customer ID identifying the consumer along with the current session ID. This information can be stored into a special lookup table that can be used by the agent to identify the customer. At step **908**, at terminal **104N**, the agent types the current session ID into text box **804** (see FIG. **8A**).

At step **910**, in response to a loss of focus or a pressing of the Enter key, the (administration) Master Applet at terminal **104N** sends a command to WTS server **144**, to retrieve the information in participant list **1**, URL history list **1**, and data list **1** (see FIG. **6**) for the Agent Applet.

At step **912**, WTS server **144** sends the information requested to the Agent Applet (via the Master Applet).

At step **914**, the Agent Applet at terminal **104N** displays some information from participant list **1** and URL history list **1** in (participant) scrollable list box **816** and (URL history) scrollable list box **818**, respectively.

At step **916**, the agent selects join button **806** in agent session interface **800A** displayed on terminal **104N**.

At step **918**, in response to the selection at step **916**, through the socket connection which has been established for agent session interface displayed on terminal **104N**, the (administration) Master Applet sends WTS server **144** a command to join the selected session. Based on the identification associated with the socket connection, WTS server is able to generate a ParticipantID for browser **114N** and to find the ParticipantAddress for terminal **104N**.

At step **920**, WTS server **144** stores the ParticipantID and ParticipantAddress into participant list **1**. At this step, participant list **1** includes two participant records (two rows) containing the PaticipantIDs for browsers **114A** and **114N** respectively.

At step **922**, at terminal **104N**, the agent selects: leading check box **810** or following check box **812**, or both of them. By only selecting leader check box **810**, the activities at terminal **104N** are synchronized at terminal **104A**, but not other way around. By only selecting follower check box **812**, the activities at terminal **104A** are synchronized at terminal **104N**, but not other way around. By selecting both leader and follower check boxes **810** and **812**, the activities at terminals **104A** and **104N** are synchronized with each other (bi-directional synchronization). In response to the selection(s), through the socket connection which has been established for agent session interface **800A**, the (administration) Master Applet sends WTS server **144** a command designating the synchronization direction. WTS server **144** stores the synchronization direction information into the Direction fields of the two records in participation list **1**. In this example, it is assumed that the bi-directional synchronization has been selected for terminals **104A** and **104N**.

At step **924**, WTS server **144** sends the (administration) Master Applet the URL of the web page being currently browsed at terminal **104A**.

At step **926**, the Agent Applet at terminal **104N** opens a browser window **1004** (a second browser instance) as shown in FIG. **10**.

At step **928**, browser **114N** downloads the web page identified by the URL from consumer page repository **146**, and displays it in browser window **1004**. A (consumer) Master Applet, a set of DTS Applets, and a SessionID Applet are embedded in the web page downloaded.

At step **930**, browser **114N** downloads (consumer) Master Applet **124N**, set of DTS Applets **126N**, and SessionID Applet **128N**.

At step **932**, the web pages displayed in second browser window **1004** at terminal **104N** are being synchronized with the web pages being displayed at terminal **104A**.

After step **932**, if the agent (the first agent) at terminal **104A** needs assistance from another agent (the second agent) at terminal **104K**, the first agent can call the second agent and tell him/her the current session ID. The second agent can then join the current session using an agent session interface as shown in FIG. **8A** displayed at terminal **104K**.

Referring to FIG. **10**, there is shown a screen display containing two browser instances (**800A** and **1004**) at terminal **104N**, in accordance with the present invention.

As shown in FIG. **10**, at terminal **104N**, the first browser instance provides an agent session interface **800A** to control and monitor the current session, and the (administration) Master Applet for agent session interface **800A** establishes and maintains a socket connection for agent session interface **800A**. The second browser instance provides a browser window **1004** to display the web pages being synchronized. (Consumer) Master Applet **124N** establishes and maintains a socket connection for each web page displayed in browser window **1004**.

Referring to FIG. **11**, there is shown a flowchart illustrating the operation of web page synchronization, in accordance with the present invention.

In the example shown in FIG. **11**, it is assumed that: (1) a consumer at terminal **104A** is browsing web pages from consumer page repository **146** via browser **114A**, (2) a session has been created for browser **114A**, (3) session list **1** and participant list **1** shown in FIG. **6** have been created for the session, (4) bi-directional synchronization has been selected for terminal **104A** and all participant terminals, and (5) the (consumer) Master Applet, DTS Applets, and SessionID Applet have been downloaded into browser **104A** and all participant browsers.

As shown in FIG. **11**, at step **1104**, browser **114A** loads a web page either from consumer page repository **146** or from memory area **115A** in terminal **104A**. If Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A** had not been download to browser **114A**, browser **114A** would download these Applets from consumer page repository **146**. However, in this example, these Applets are assumed to be downloaded.

At step **1106**, in response to the loading of the web page, browser **114A** initializes and invokes Master Applet **124A**, DTS Applets **126A**, and SessionID Applet **128A**.

At step **1108**, Master Applet **124A**: (1) opens a dedicated socket, and establishes a socket connection to WTS gateway **142** for browser **114A** and the web page loaded, and (2) via the socket connection, sends WTS server **144** a command together with an ID unique to browser **114A** and the URL of the web page loaded. Based on the unique ID, WTS server is able to identify the session created for browser **114A**.

At step **1110**, WTS server **144** identifies the session for browser **114A**.

At step **1112**, WTS server **144** locates all IP addresses assigned to participant terminals in participant list **1** (shown in FIG. **6**), and sends a command, together with the URL, to all the participant terminals (except that WTS server **144** does not sent the URL to terminal **104A**, because the URL is originated from terminal **104A**).

At step **1114**, upon receiving the command, the (consumer) Master Applets in the participant terminals initialize themselves and pass the URL to their respective browsers.

At step **1116**, the respective browsers in the participant terminals download and display the web page according to the URL.

It should be noted that, like terminal **104A**, each of the participant terminals (at which agent session interface is displayed) can lead the page synchronization using the operation shown in FIG. **11**.

Referring to FIG. **12A**, there is shown a web page **1200** containing five data fields, specifically: name **1202**, time period **1204**, account balance **1206**, payment **1208**, comments **1210**, a text box **1212** for displaying the current session ID, and a text box for displaying the call center number the consumer can call, in accordance with the present invention.

Referring to FIG. **12B**, there is shown a web page that is similar to that of FIG. **12A**, except that the data in the name field **202** has been changed from Susan King to Sue Grant, and a comment indicating that the name contained in field **202** has been changed is provided in comments field **1210**. These changes are also displayed in a web page **1200'** synchronized at a participant terminal, in accordance with the present invention.

Referring to FIG. **13**, there is shown a flowchart illustrating the operation of data synchronization, in accordance with the present invention.

In the example shown in FIG. 13, it is assumed that: (1) a customer at terminal 104A is browsing web pages via browser 114A, (2) a session has been created for terminal 104A, (3) session list 1 and participant list 1 shown in FIG. 6 has been created for the session, (4) terminal 104N is one of the participants, (5) web page 1200 containing five data fields shown in FIG. 12A is displayed on terminals 104A and all participant terminals, (6) a bi-directional synchronization has been selected for terminal 104A and all participant terminals, (7) the (consumer) Master Applet, DTS Applets, and SessionID Applet have been downloaded to browser 114A and all participant browser, (8) the DTS Applets contains five individual Applets: DTS $Applet_1$, DTS $Applet_2$, DTS $Applet_3$, DTS $Applets_4$, and DTS $Applet_5$, (9) these five individual DTS Applets are respectively responsible for monitoring and processing the events occurred on the five data fields of web page 1200 shown in FIG. 12A, (10) (consumer) Master Applet 124A has established a dedicated socket connection to WTS gateway 142 for web page 12A displayed at terminal 104A, and (11) the customer at terminal 104A wants to make changes to name field 1202 from Susan King to Sue Grant.

As shown in FIG. 13, at step 1304, the customer changes the name in name field 1202 from Susan King to Sue Grant.

At step 1306, in response to a loss of focus on name field 1202 or pressing the Enter key, DTS Applet, detects the change and passes the change to Master Applet 124A. Note that the change of the form field value could also be detected through a proprietary interface between the master applet and the web browser.

At step 1308, via the dedicated socket connection, Master Applet 124A sends WTS server 144 a command specifying the change of name field 1202. Since this change is passed to WTS server 144 via the dedicated socket connection established for web page 1200, WTS server 144 is able to recognize the origin of the command, web page 1200, and the name field upon which the change was made.

At step 1310, WTS server 144 identifies the session created for browser 114A.

At step 1312, WTS server 144 locates the IP addresses assigned to participant browsers in participant list 1 and sends a command (together with the change of name field 1202) to the Master Applets in all other participant terminals (except that WTS server 144 does not send the command and change to browser 114A, since this change originated from browser 114A).

At step 1314, upon receiving the command, the (consumer) Master Applets (including Master Applets 124N) pass the change of name field 1200 to their respective DTS Applets, including the DTS $Applet_1$ at browser 114N.

At step 1316, the DTS Applet, display the update "Susan Grant" into the name fields on respective web page 1200 displayed on the respective terminals, including terminal 104N.

It should be noted that the operation shown in FIG. 13 can be used to perform data synchronization for the other four data fields on web page 1200 shown in FIG. 12A.

It should also be noted that the data field synchronization can also be performed at terminal 104N. For example, as shown in FIG. 12B, when the agent at terminal 104N enters comments of "Account's name had been changed" to comments field 1210' on web page 1200', this updates will be displayed in comments field 1210 at terminal 104A, by using the operation shown in FIG. 13.

Referring to FIG. 14, there is shown a flowchart illustrating the operation of web page tracking, in accordance with the present invention.

In the example shown in FIG. 14, it is assumed that: (1) a customer at terminal 104A is browsing web pages via browser 114A, (2) a session has been created for terminal 104A and all participant terminals, (3) session list 1, participant list 1, and URL history list 1 shown in FIG. 6 have been created for the session, (4) bi-directional synchronization has been selected for terminal 104A and all participant terminals, and (5) the (consumer) Master Applet, DTS Applets, and SessionID Applet have been downloaded into terminals 104A and all participant terminals.

As shown in FIG. 14, at step 1404, browser 114A downloads a web page from either the consumer page repository 146 or memory area 115A of terminal 104A. If Master Applet 124A, DTS Applets 126A, and SessionID Applet 128A had not been download to terminal 104A, browser 114A would download them from HTTP server 152. However, in this example, these Applets have been downloaded.

At step 1406, web browser 114A initializes and invokes Master Applet 124A, DTS Applets 126A, and SessionID Applet 128A.

At step 1408, Master Applet 124A opens a dedicated socket and establishes a socket connection to WTS gateway 142 for web browser 114A and the web page loaded. Master Applet 124A then sends WTS server 144 a command, together with: (1) an ID unique to browser 114A, and (2) the URL of the web page loaded. When commands and URL are delivered through this socket connection, WTS server 144 is able to recognize the origin of the commands and URL.

At step 1410, WTS server 144 identifies the session ID for browser 114A.

At step 1412, WTS server 144 locates the session list 1 and URL history list 1.

At step 1414, WTS server 144 issues a time stamp (loading time) for indicating the time at which the command was received, and stores the URL and time stamp to URL history list 1.

At step 1416, browser 114A sends WTS server 144 a request to load a subsequent web page.

At step 1418, before loading the subsequent web page, via the socket connection, Master Applet 124A sends WTS server 144 a command, together with the URL, to inform WTS server 144 that the current web page has been unloaded.

At step 1420, WTS server 144 identifies the session for terminal 104A.

At step 1422, WTS server 144 locates the session list 1 and URL history list 1.

At step 1424, WTS server 144 issues a time stamp (unloading time) for indicating the time at which the command was received, and stores the URL and time stamp to URL history list 1.

At step 1426, Master Applet 124A disconnects the socket connection for the web page that has been unloaded.

Referring to FIG. 15, there is shown a flowchart illustrating the operation of data tracking, in accordance with the present invention.

In the example shown in FIG. 15, it is assumed that: (1) a customer at terminal 104A is browsing web pages via browser 114A, (2) a session has been created for terminal 104A, (3) session list 1 and participant list 1 shown in FIG. 6 has been created for the session, (4) terminal 104N is one of the participants, (5) web page 1200 containing five data fields shown in FIG. 12A is displayed on terminals 104A and all participant terminals, (6) a bi-directional synchronization has been selected for terminal 104A and all participant terminals, (7) the (consumer) Master Applet, DTS Applets, and SessionID Applet are downloaded to terminal 104A and all participant terminals, (8) the DTS Applets contains five individual Applets: DTS $Applet_1$, DTS $Applet_2$, DTS $Applet_3$, DTS $Applets_4$, and DTS $Applet_5$, (9) these five individual DTS Applets are respectively responsible for

displaying, monitoring and processing the events occurred on the five data fields of web page **1200** shown in FIG. **12**A, (10) Master Applet **124**A has established a dedicated socket connection to WTS server **144** for web page **1200** displayed on terminal **104**A, and (11) the customer at terminal **104**A wants to make changes to name field **1202** from Susan King to Sue Grant.

As shown in FIG. **15**, at step **1504**, the customer changes the name in name field **1502** from Susan King to Sue Grant.

At step **1506**, in response to a loss of focus on name field **1202** or pressing the Enter key, DTS Applet, detects the change and passes the change to Master Applet **124**A.

At step **1508**, via the dedicated socket connection, Master Applet **124**A sends WTS server **144** a command together with the change of name field **1202**. Since this change is passed to WTS server **144** via the dedicated socket connection established for web page **1200**, WTS server **144** is able to recognize the origin of the command, web page **1200**, and the name field upon which the change was made.

At step **1510**, WTS server **144** identifies the session created for terminal **104**A.

At step **1512**, WTS server **144** stores the URL and update of name field **1202** into data list **1**.

It should be noted that the operation shown in FIG. **15** can be used to perform data tracking for the other four data fields on web page **1200**, and to perform data tracking for all participant terminals.

Referring to FIG. **16**, there is shown a flowchart illustrating the operation of repeating browsing activities previously performed to a web page in a session, in accordance with the present invention.

In the example shown in FIG. **16**, it is assumed that: (1) a user at terminal **104**A had previously browsed web pages from consumer page repository **146** via browser **114**A, (2) a session list 1 shown in FIG. **6** was created for browser **114**A, (3) an agent is on duty at terminal **104**N in a call center, and agent class has been assigned to browser **104**N, (4) at browser **114**N, the first and second browser instances for the agent as shown in FIG. **10** have been displayed, (5) via respective socket connections established by Master Applet **124**N, the first and second browser instances for the agent as shown in FIG. **10** have been connected to WTS gateway **142**, (6) Master Applets (**124**A and **124**N), DTS Applets (**126**A and **126**N) and SessionID Applets (**128**A and **128**N) have been downloaded into terminals **104**A and **104**N respectively, (7) the agent has selected and joined the session created for browser **114**A, (8) at browser **114**N, the second browser instance for the agent as shown in FIG. **10** is being synchronized with browser **114**A, and (9) bi-direction synchronization has been selected for browsers **114**A and **114**N.

At step **1602**, the agent selects a session and reviews the URLs for all the web pages previously browsed by browser **114**A in the selected session.

At step **1604**, to display an individual web page previously browsed by browser **114**A, the agent selects a URL from scrollable list box **818** (or scrollable list box **858**) and double-clicks on it.

At step **1606**, agent Master Applet **124**N sends WTS server **144** a command together with the selected URL, via its respective socket connection.

At **1608**, upon receiving the command, WTS server **144** retrieves the web page identified by the URL from HTTP server **152** (see FIG. **2**). If the user browser previously performed any changes to the data fields of the web page, these changes are stored in data list **1** as shown in FIG. **6**.

At **1610**, WTS server **144** sends the web page to user and agent Master Applets **124**A and **124**N. If the user browser previously performed some changes to the data fields of the web page, WTS server **114** also sends these changes to user and agent Master Applets **124**A and **124**N.

At step **1614**, agent and user Master Applets **124**A and **124**N instruct their respective browsers to display the web page previously browsed. If the user browser previously performed any changes to the data fields of the web page, the user and agent Master Applets receive the final data values of the changes and cause each data filed to display these changes on the web page.

Referring to FIG. **17**, there is shown a flowchart illustrating the operation of replaying a browsing session, in accordance with the present invention.

In the example shown in FIG. **17**, it is assumed that: (1) a user at terminal **104**A had browsed web pages from consumer page repository **146** via browser **114**A, (2) a session list 1 shown in FIG. **6** was created for browser **114**A, (3) an agent is on duty at terminal **104**N in a call center, and agent class has been assigned to browser **104**N, (4) at browser **114**N, the first and second browser instances for the agent as shown in FIG. **10** have been displayed, (5) via respective socket connections established by Master Applet **114**N, the first and second browser instances for the agent as shown in FIG. **10** have been connected to WTS gateway **142**, (6) Master Applets (**124**A and **124**N), DTS Applets (**126**A and **126**N) and SessionID Applets (**128**A and **128**N) have been downloaded into terminals **104**A and **104**N respectively, (7) the agent has selected and joined the session created for browser **114**A, (8) at browser **114**N, the second browser instance for the agent as shown in FIG. **10** is being synchronized with browser **114**A, and (9) bi-direction synchronization has been selected for browsers **114**A and **114**N.

At step **1702**, the agent selects a session and reviews the URLs for all the web pages previously browsed by browser **114**A in the selected session.

At step **1704**, to re-browse all web pages previously browsed by browser **114**A, the agent selects Session Replay button **820** in the agent session interface as shown in FIG. **10**.

At step **1706**, agent Master Applet **124**N sends WTS server **144** a command together with the session selected, via its respective socket connection.

At **1708**, upon receiving the command, WTS server **144** retrieves the web pages identified by the URLs in the selected session from HTTP server **152** (see FIG. **2**). If the user browser previously performed any changes to the data fields of the web pages, these changes are stored in data list **1** as shown in FIG. **6**.

At **1710**, WTS server **144** sequentially sends the web pages to user and agent Master Applets **124**A and **124**N, in accordance with the loading and unloading times stored in URL history list **1** (see FIG. **6**). If the user browser previously performed any changes to the data fields of the web pages, WTS server **144** also sends these changes to user and agent Master Applets **124**A and **124**N, in accordance with the loading and unloading times stored in data list **1**.

At **1712**, user and agent Master Applets **124**A and **124**N instruct their respective browsers **114**A and **114**N to displayed the web pages sent from WTS server **144**. If the user browser previously performed any changes to the web pages to the data fields of the web pages, user and agent Master Applet **124**A and **124**N receive the final data values of the changes and cause each data field to display these changes on the web pages.

Since the loading time and unloading time of the URLs and the setting time for data fields are recorded in URL history list **1** and data list **1**, if desired, all the web pages identified by the URLs and the activities performed to the data fields can be duplicated (loading the web page, setting data fields on the web page, and unloading the web page) according to the timing information. If desired, the sequence of URLs and data entry may be played proportionally faster or slower than the original session.

It should be noted that, in the above-described embodiments, all the Applets (Master Applets, DTS Applets, SessionID Applets, and Agent Applet) embedded into web

pages are written using Java. However, some or all of these Applets can be written using a browser script language, such as Java Script. More specifically, some of the features provided by these Applets can be directly written into web pages using the browser script language, instead of using applet tags to link these Applets. When a web browser downloads a web page containing the Applets written in browser script language, it stores these Applets into the memory area of the terminal on which the web browser is running, and then initializes and invokes them. It should also be noted that a session could conceivably be replayed directly from database **156** in FIG. **1**. In this scenario, the session has been terminated, but because all of the necessary data was stored in a database prior to the session being purged, an active session is not required to implement playback.

It can thus be seen that there has been provided by the present invention a new and useful mechanism to record the detailed browsing activities of an individual browser, and to reproduce a recorded set of browser activities at one or more browsers. The invention described herein has utility in a variety of applications including:

Consumer behavior analysis. The present invention can assist web developers or other interested parties in analyzing an individual consumer's web browsing activities. This function is comparable to the type of consumer behavior analysis currently performed in retail stores when analyzing the specific actions of a consumer as he or she walks through a store making selections.

Quality assurance. Currently telephone conversations between customers and agents in call centers are recorded for quality assurance purposes. Supervisors are thereafter able to listen to the recorded conversations to insure that customers are treated properly and that agents are following the appropriate procedures. When web conferencing is used in concert with a call center, it may be appropriate to record the synchronized browser activity between the call center agent and the end consumer. This recorded web session could be played back in synchronization with the voice record of the call.

Training. The present invention enables the presentation of a recorded sequence of web pages to a group of individuals, each utilizing a separate web browser. The ability to control the rate of playback would allow an instructor to slow down the presentation to answer questions, or to pause the presentation to take a break, provide comment or answer questions.

While the invention has been illustrated and described in detail in the drawing and foregoing description, it should be understood that the invention may be implemented through alternative embodiments within the spirit of the present invention. Thus, the scope of the invention is not intended to be limited to the illustration and description in this specification, but is to be defined by the appended claims.

What is claimed is:

1. A method for repeating browsing activities performed during a web browsing session using a first web browser on a first client computer, said browsing activities including a series of requests for retrieving web pages from a web server, the method comprising the steps of:

(a) receiving at said web server an initial request from said first web browser for retrieving an initial page, said initial request indicating the beginning of said web browsing session;

(b) creating a session file at said web server for said web browsing session in response to said initial request;

(c) recording said browsing activities performed during said web browsing session to said session file;

(d) assigning a unique session ID to said session file;

(e) retrieving the browsing activities recorded during said web browsing session to said session file from said web server to a second web browser in response to a request from said second web browser identifying said session file by said session ID; and

(f) repeating the recorded browsing activities at said second web browser; and wherein

said web server includes multiple session files, each one of said multiple session files identified by a unique session ID, said multiple session files corresponding to multiple web browsing sessions occurring at said first client computer.

2. The method according to claim **1**, further comprising the steps of:

(d) retrieving the recorded browsing activities to a third web browser;

(e) repeating the recorded browsing activities at said third web browser; and

(f) synchronizing the browsing activities that are being repeated at said second and third web browsers.

3. The method according to claim **2**, wherein:

said step of recording browsing activities performed using said first web browser includes the step of recording a time reference for said browsing activities; and

said step of repeating the recorded browsing activities at said second web browser includes the step of repeating browsing activities according to said time references.

4. In a computer network including a network server and at least one client computer, a method for repeating browsing activities occurring during multiple web browsing sessions on said at least one client computer, the method comprising the steps of:

(a) creating a plurality of session files at said network server, each of said session files being associated with a respective one of said multiple web browsing sessions;

(b) recording the browsing activities for each of the multiple web browsing sessions into associated session files at the network server;

(c) assigning a unique session ID to each one of said session files;

(d) selecting one of said session files for replay, said selected session file being identified for selection by its assigned session ID;

(e) retrieving the selected session files to an administrative web browser; and

(f) repeating the recorded browsing activities associated with said selected session file at said administrative web browser.

5. The method according to claim **4**, wherein:

said step of recording the browsing activities for each of said web browsing sessions into associated session files at the network server includes the step of recording a time reference for said browsing activities with each session file; and

said step of repeating the recorded browsing activities associated with said selected session file at said administrative web browser includes the step of repeating browsing activities according to the time reference associated with said selected session file.

* * * * *

US006549944B1

(12) **United States Patent**
Weinberg et al.

(10) **Patent No.: US 6,549,944 B1**
(45) **Date of Patent: *Apr. 15, 2003**

(54) **USE OF SERVER ACCESS LOGS TO GENERATE SCRIPTS AND SCENARIOS FOR EXERCISING AND EVALUATING PERFORMANCE OF WEB SITES**

(75) Inventors: **Amir Weinberg**, Zoran (IL); **Eduardo Alperin**, Ramat-Gan (IL)

(73) Assignee: **Mercury Interactive Corporation**, Sunnyvale, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 108 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **09/610,909**

(22) Filed: **Jul. 6, 2000**

Related U.S. Application Data

(63) Continuation of application No. 09/315,795, filed on May 21, 1999, which is a continuation of application No. 08/949,680, filed on Oct. 14, 1997, now Pat. No. 5,974,572, which is a continuation-in-part of application No. 08/840,103, filed on Apr. 11, 1997, now Pat. No. 5,870,559.

(60) Provisional application No. 60/028,474, filed on Oct. 15, 1996.

(51) **Int. Cl.**[7] **G06F 15/173**

(52) **U.S. Cl.** **709/224**; 709/200; 709/203; 714/47; 714/33

(58) **Field of Search** 709/201, 217, 709/219, 224, 228; 707/10, 102; 705/14; 713/200; 714/1, 25, 31, 33; 345/357

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,303,166	A		4/1994	Amalfitano et al.
5,446,874	A	*	8/1995	Waclawsky et al. 714/1
5,537,542	A	*	7/1996	Eilert et al. 709/201
5,544,310	A	*	8/1996	Forman et al. 714/31
5,657,438	A		8/1997	Wygodny et al.
5,696,701	A	*	12/1997	Burgess et al. 714/25
5,761,673	A	*	6/1998	Bookman et al. 707/104
5,774,123	A	*	6/1998	Matson 345/357
5,787,254	A	*	7/1998	Maddalozzo, Jr. et al. . 709/228
5,812,780	A		9/1998	Chen et al.
5,819,066	A	*	10/1998	Bromberg et al. 707/102
5,848,415	A	*	12/1998	Guck 707/10
5,892,917	A	*	4/1999	Myerson 709/224
5,937,390	A	*	8/1999	Hyodo 705/14
5,974,572	A	*	10/1999	Weinberg et al. 714/33
6,122,740	A	*	9/2000	Andersen 713/200

OTHER PUBLICATIONS

Measuring the Performance of HTTP Daemons by Robert E. McGrath, Feb. 5, 1996.*

(List continued on next page.)

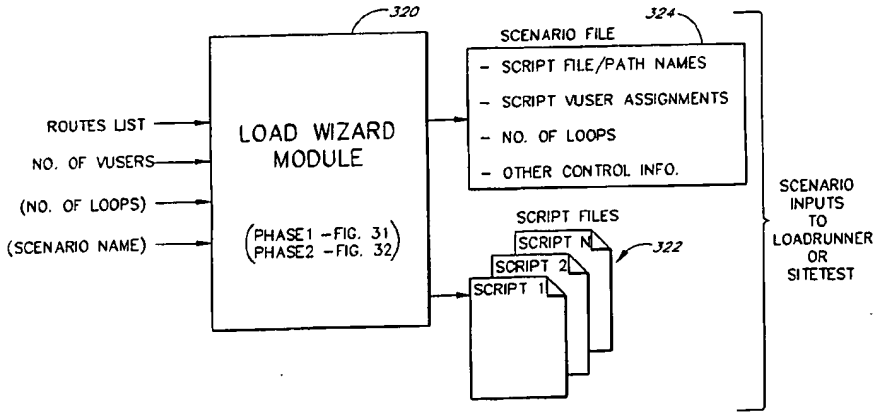
Primary Examiner—Ayaz Sheikh
Assistant Examiner—Thu Ha Nguyen
(74) *Attorney, Agent, or Firm*—Knobbe, Martens, Olson & Bear LLP

(57) **ABSTRACT**

A visual Web site analysis program, implemented as a collection of software components, provides a variety of features for facilitating the analysis, management and load-testing of Web sites. A mapping component scans a Web site over a network connection and builds a site map which graphically depicts the URLs and links of the site. Site maps are generated using a unique layout and display methodology which allows the user to visualize the overall architecture of the Web site. Various map navigation and URL filtering features are provided to facilitate the task of identifying and repairing common Web site problems, such as links to missing URLs. A dynamic page scan feature enables the user to include dynamically-generated Web pages within the site map by capturing the output of a standard Web browser when a form is submitted by the user, and then automatically resubmitting this output during subsequent mappings of the site. An Action Tracker module detects user activity and behavioral data (link activity levels, common site entry and exit points, etc.) from server log files and then superimposes such data onto the site map. A Load Wizard module uses this activity data to generate testing scenarios for load testing the Web site.

32 Claims, 32 Drawing Sheets

Microfiche Appendix Included
(1 Microfiche, 51 Pages)



OTHER PUBLICATIONS

Estimating server performance by Mari Korkea-aho, Jan. 19, 1996.*

Original "Loadrunner DB Virtual User" User's Guide for PC Platforms, Part II: SQL Vuser Generator, Copyright 1995-1996 by Mercury Interactive Corporation.

Original "Loadrunner Controller" User's Guide for PC Platforms Version 4.0, Copyright 1994, 1995 and 1996 by Mercury Interactive Corporation.

Original "Loadrunner GUI Virtual User" User's Guide for PC Platforms Version 4.0, Copyright 1995 and 1996 by Mercury Interactive Corporation.

Original "Loadrunner DB Virtual User" User's Guide for PC Platforms, Part I: Overview and DB Vuser Templates, Copyright 1995 and 1996 by Mercury Interactive Corporation.

Print-out of online help manual for InContext WebAnalyzer 1.0 product (taken from CD-Rom dated 1996).

User's Guide for NetCarta WebMapper 1.0 for Windows NT/95, dated 1996.

D. Beckett, "Combined Log System" Computer Networks and ISDN Systems, pp. 1089-1096, dated Apr. 10-14, 1995.

* cited by examiner

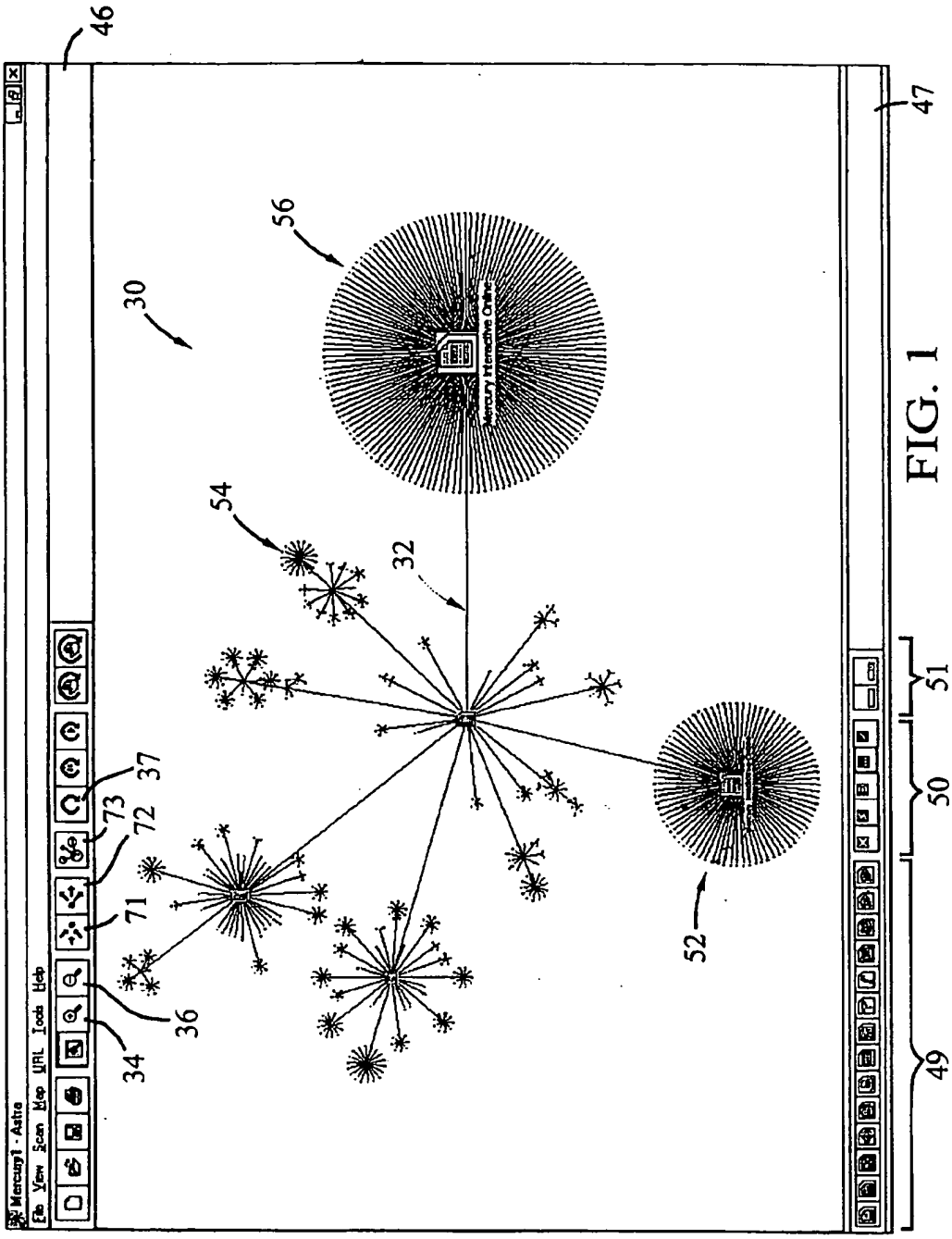

46
47
56
30
54
32
37
73
72
71
36
34
52
51
50
49
Mercury Interactive Online
Mercury1 - Astra
File View Scan Map URL Tools Help

FIG. 1

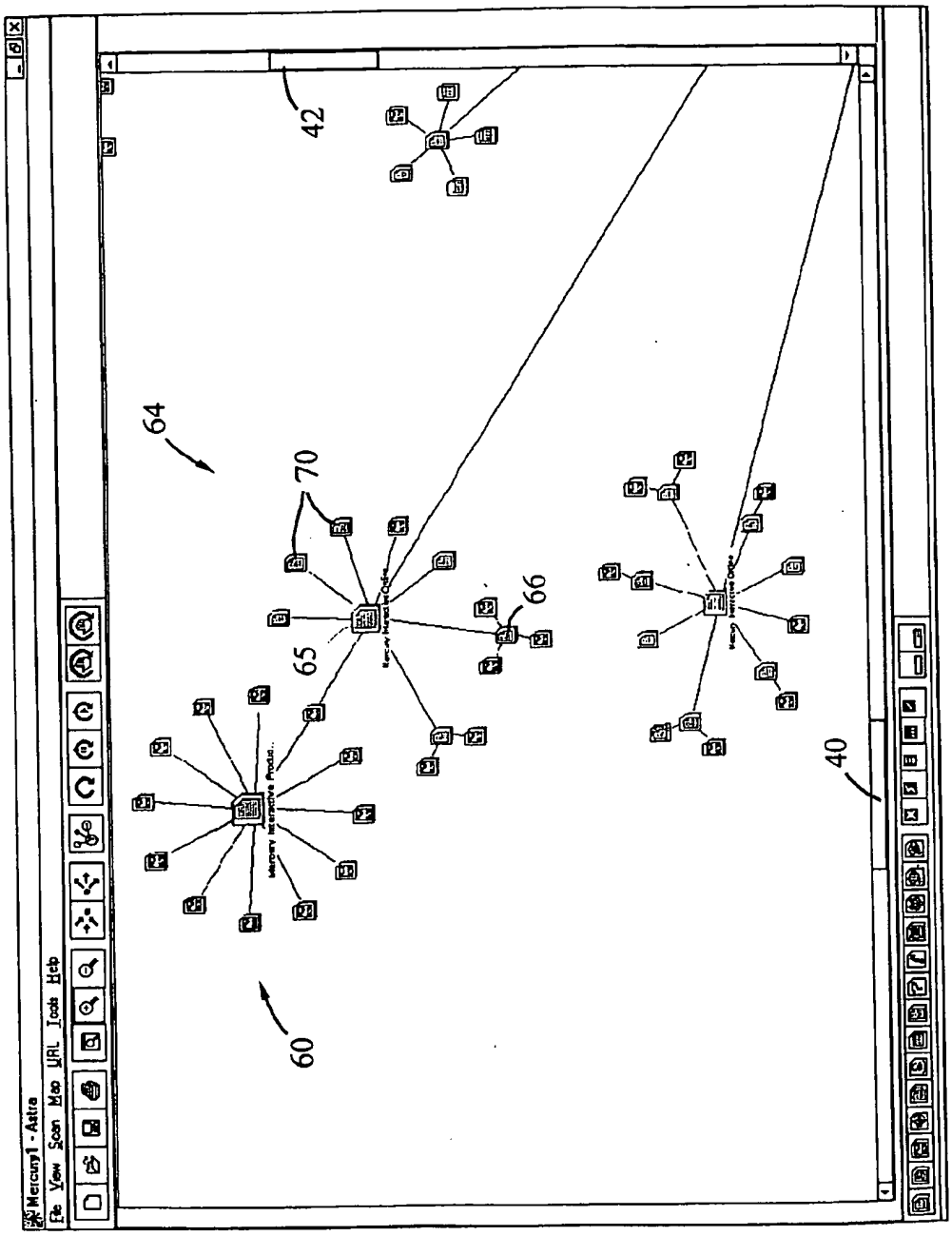

42
64
70
66
65
60
40
Mercury1 - Astra
File View Scan Map URL Tools Help

FIG. 2

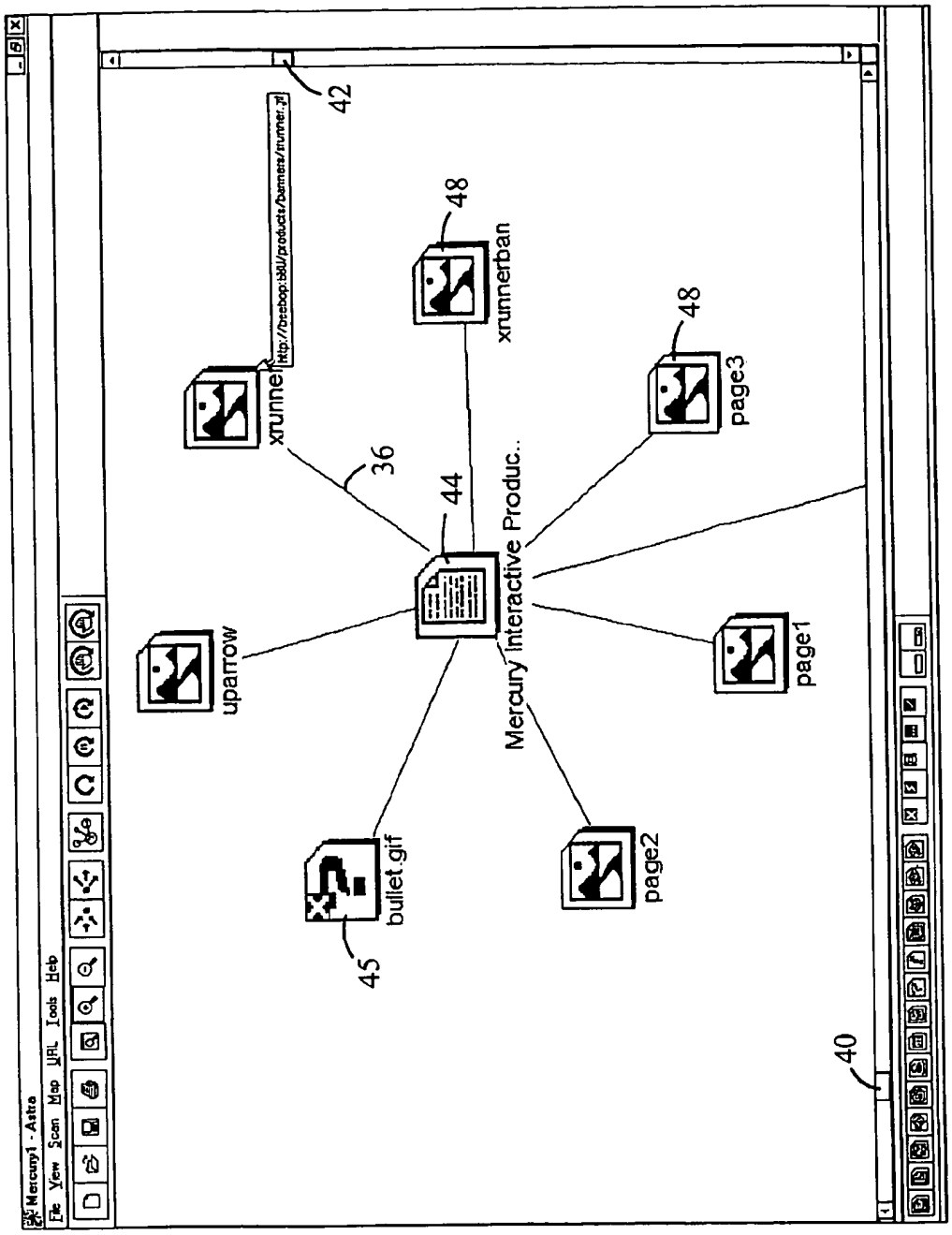

Mercury1 - Astra
File View Scan Map URL Tools Help
42
48
xrunnerban
48
page3
xrunner
36
44
Mercury Interactive Produc...
uparrow
page1
bullet.gif
page2
45
40

FIG. 3

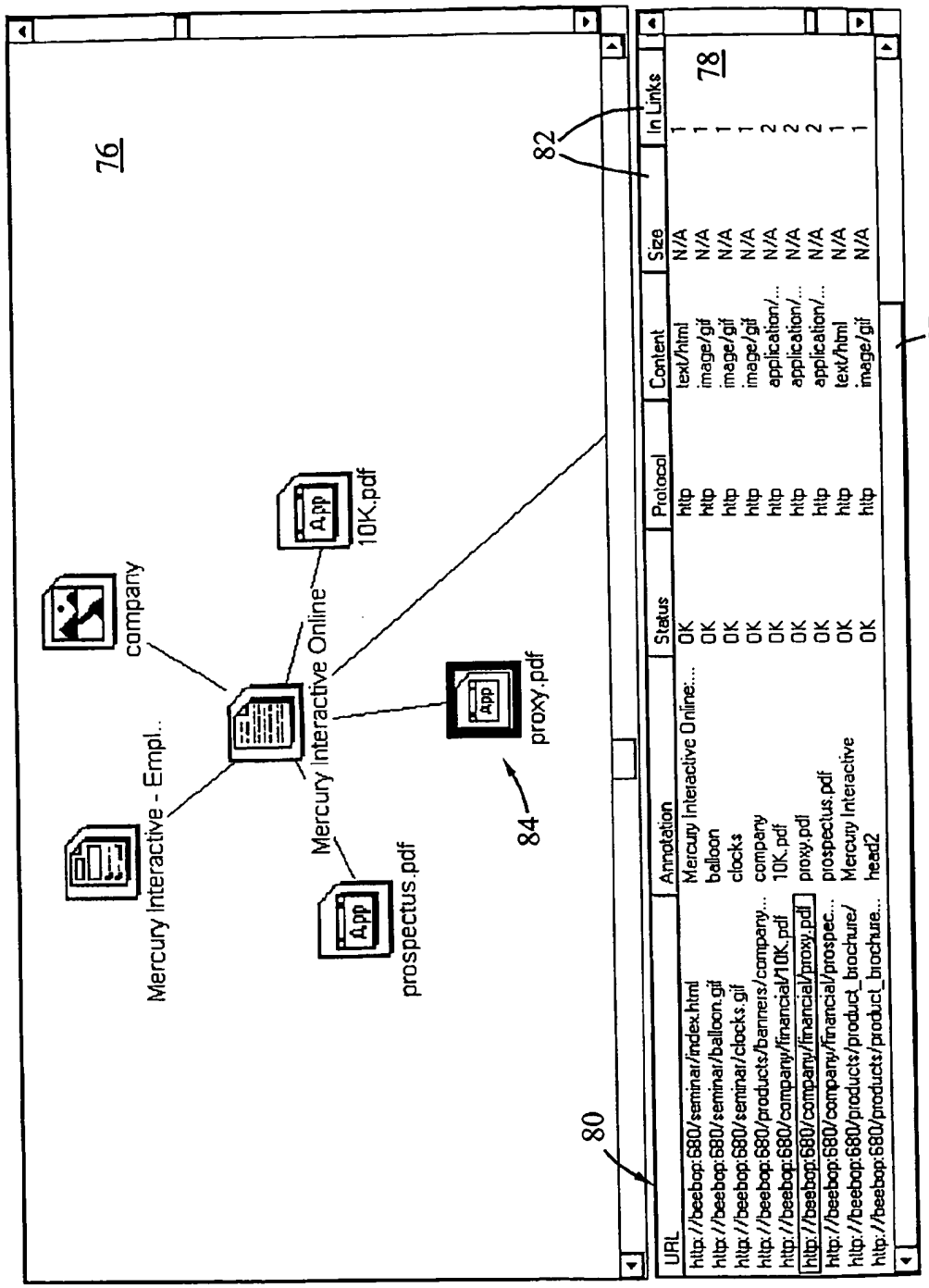

76
82
78
company
Mercury Interactive - Empl...
Mercury Interactive Online
10K.pdf
prospectus.pdf
proxy.pdf
84
80
77
URL
Annotation
Status
Protocol
Content
Size
In Links
http://beebop:680/seminar/index.html
Mercury Interactive Online:...
OK
http
text/html
N/A
1
http://beebop:680/seminar/balloon.gif
balloon
OK
http
image/gif
N/A
1
http://beebop:680/seminar/clocks.gif
clocks
OK
http
image/gif
N/A
1
http://beebop:680/products/banners/company...
company
OK
http
image/gif
N/A
1
http://beebop:680/company/financial/10K.pdf
10K.pdf
OK
http
application/...
N/A
2
http://beebop:680/company/financial/proxy.pdf
proxy.pdf
OK
http
application/...
N/A
2
http://beebop:680/company/financial/prospec...
prospectus.pdf
OK
http
application/...
N/A
2
http://beebop:680/products/product_brochure/
Mercury Interactive
OK
http
text/html
N/A
1
http://beebop:680/products/product_brochure...
head2
OK
http
image/gif
N/A
1

FIG. 4

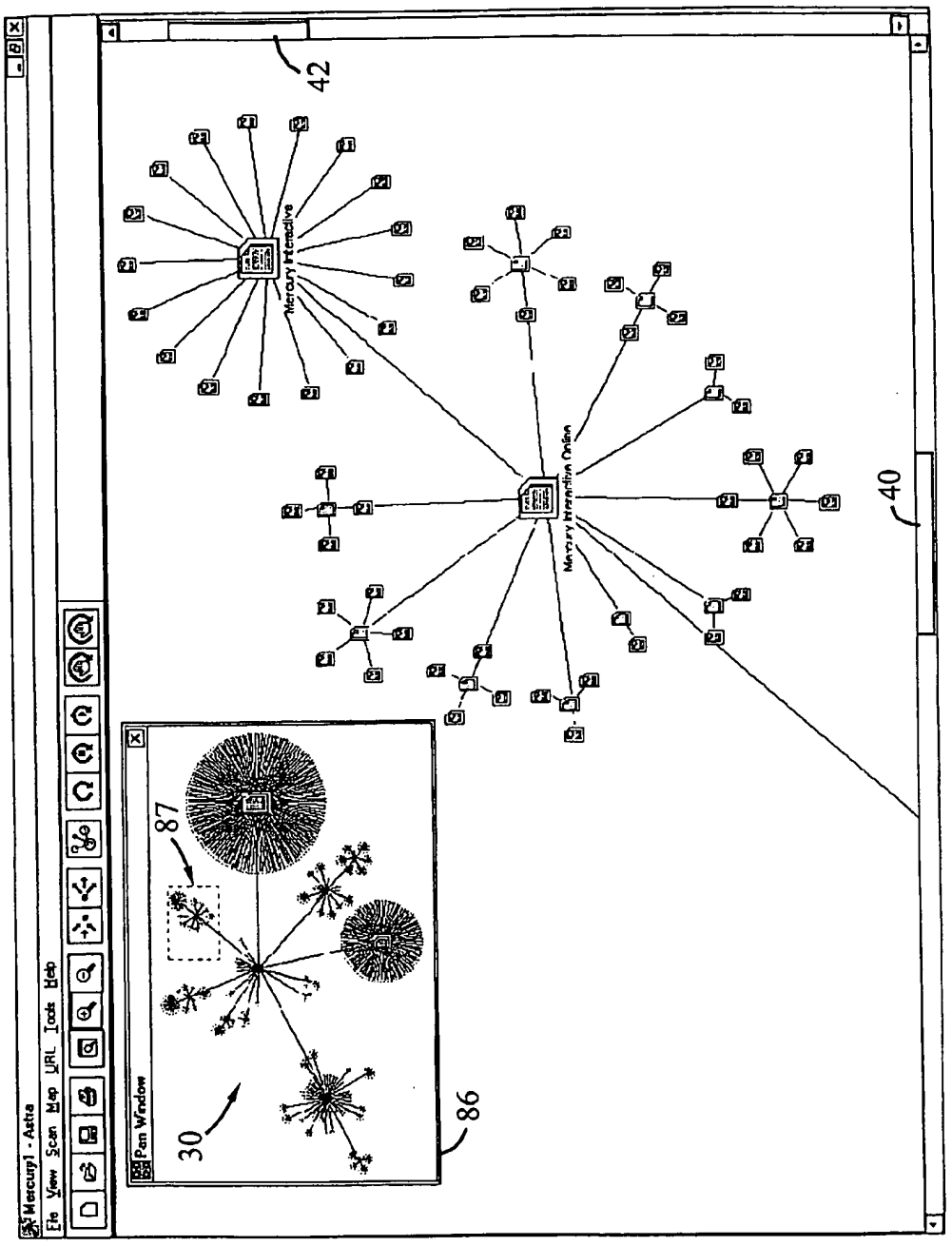

42
40
87
30
86
Pan Window
Mercury Interactive
Mercury Interactive Online
Mercury - Astra
File View Scan Map URL Tools Help

FIG. 5

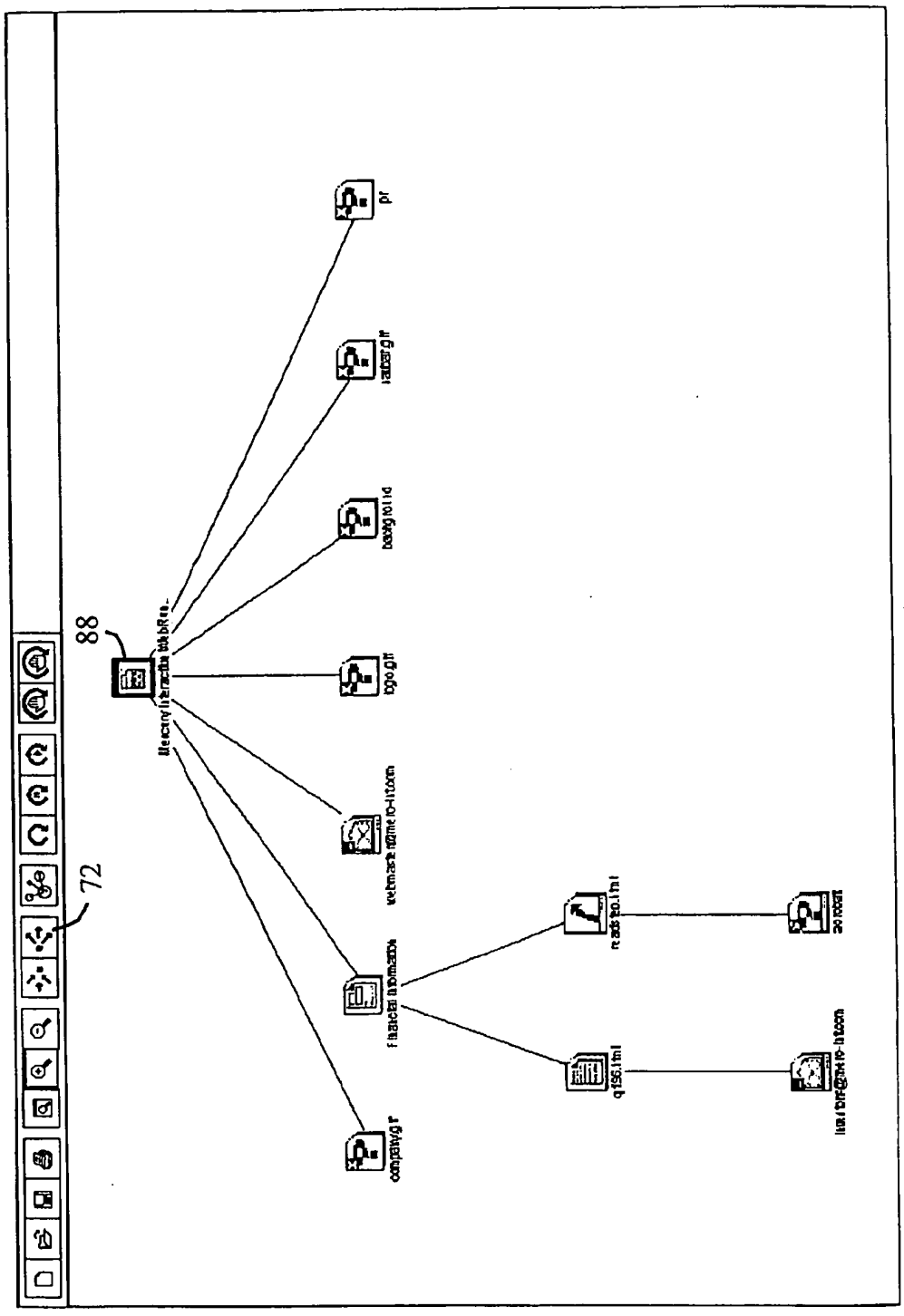
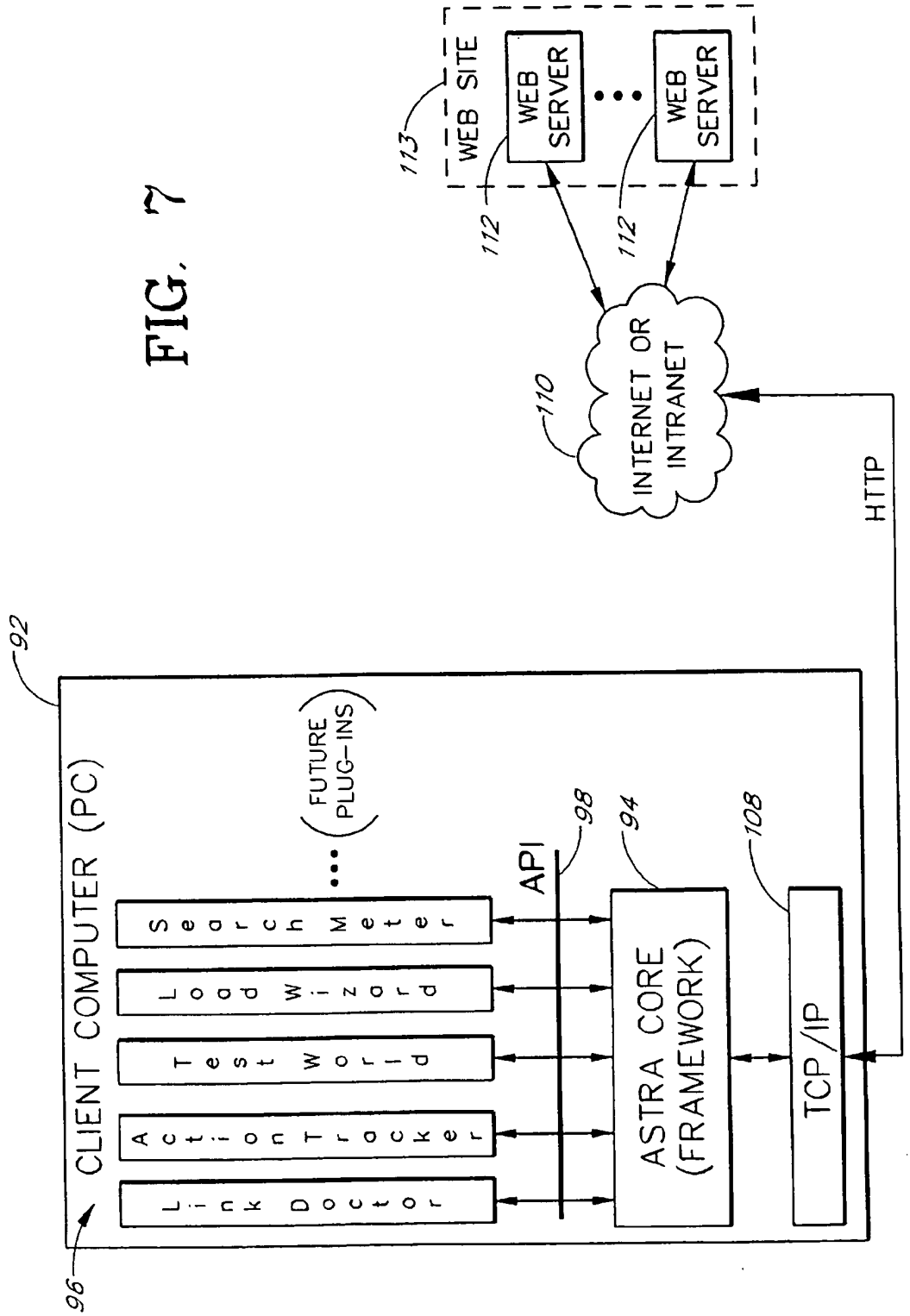


FIG. 6

FIG. 7


WEB SITE
113
WEB SERVER
WEB SERVER
112
112
INTERNET OR INTRANET
110
HTTP
92
CLIENT COMPUTER (PC)
FUTURE PLUG-INS
API
98
94
108
SearchMeter
LoadWinzorld
TestWorld
ActionTracker
Link Doctor
ASTRA CORE (FRAMEWORK)
TCP/IP
96

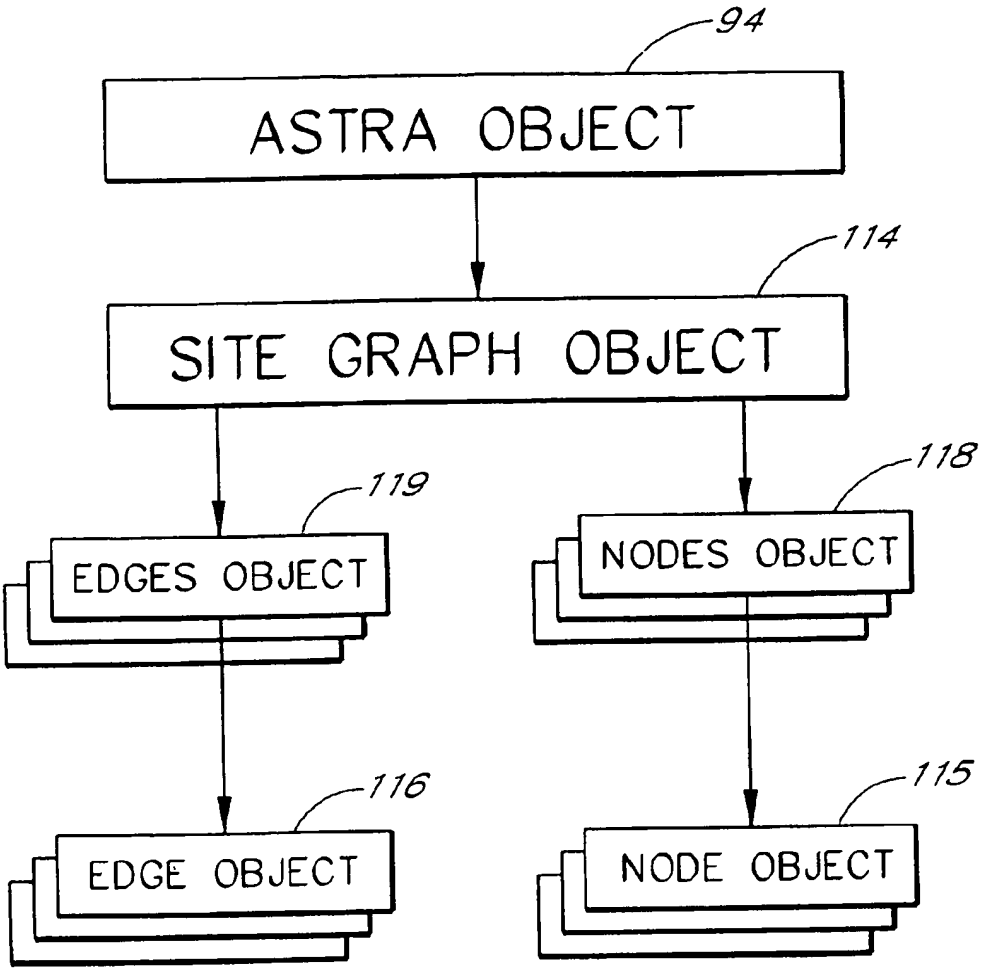

94
ASTRA OBJECT
114
SITE GRAPH OBJECT
119
EDGES OBJECT
118
NODES OBJECT
116
EDGE OBJECT
115
NODE OBJECT

FIG. 8

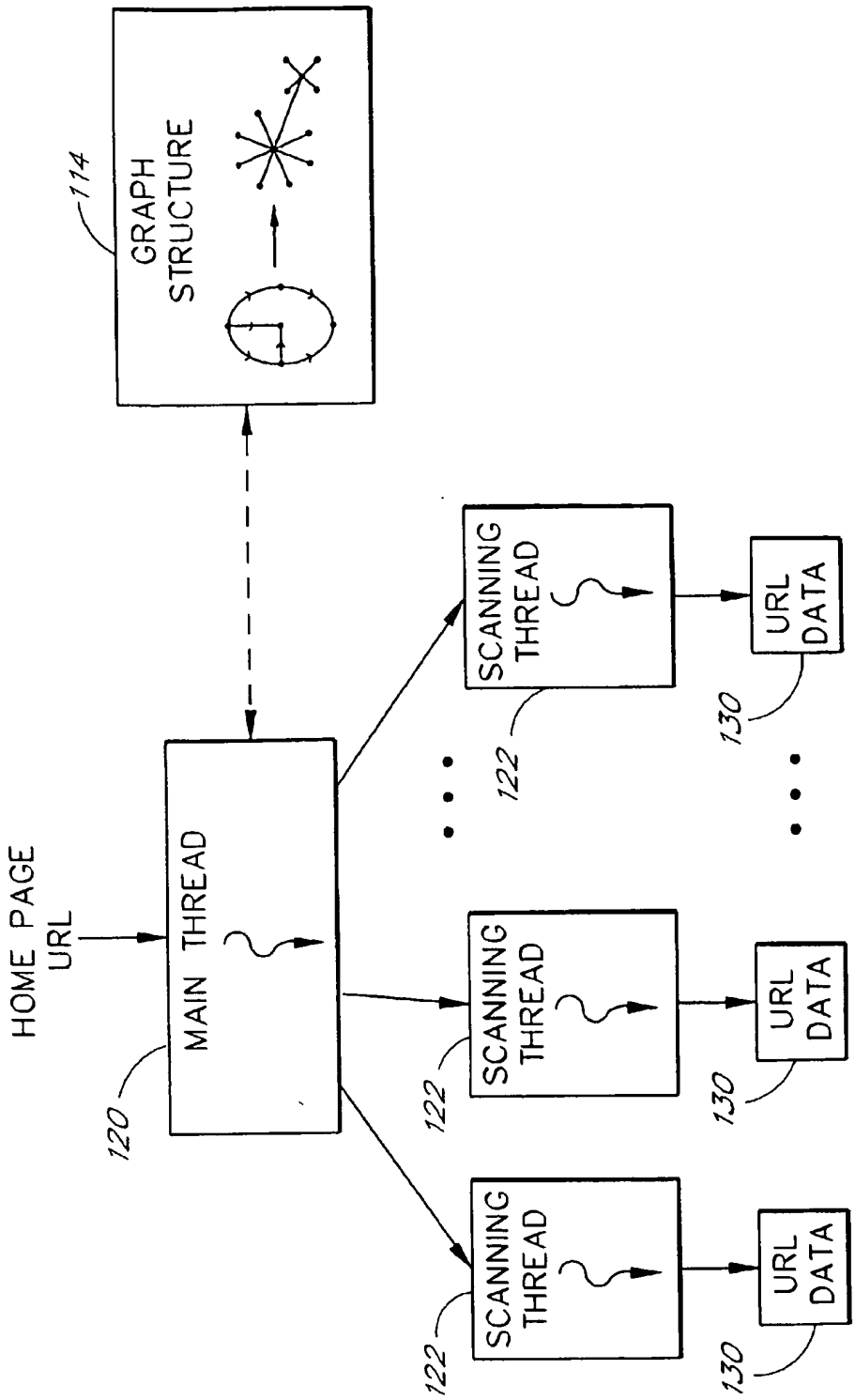

GRAPH STRUCTURE
114
HOME PAGE URL
MAIN THREAD
120
SCANNING THREAD
122
URL DATA
130

FIG. 9

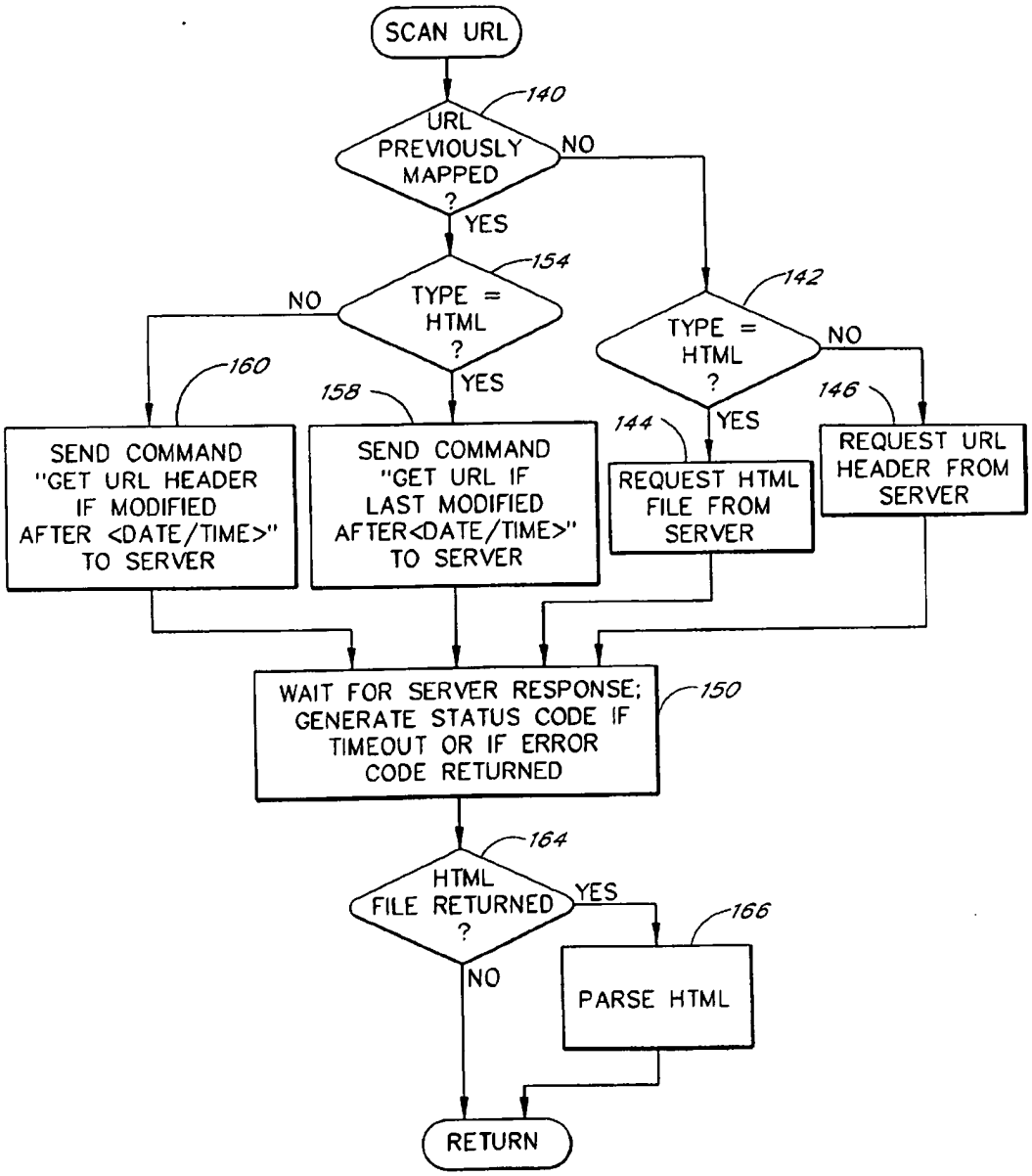

SCAN URL
140
URL PREVIOUSLY MAPPED ?
NO
YES
154
TYPE = HTML ?
NO
YES
142
TYPE = HTML ?
NO
YES
160
158
144
146
SEND COMMAND "GET URL HEADER IF MODIFIED AFTER <DATE/TIME>" TO SERVER
SEND COMMAND "GET URL IF LAST MODIFIED AFTER<DATE/TIME>" TO SERVER
REQUEST HTML FILE FROM SERVER
REQUEST URL HEADER FROM SERVER
WAIT FOR SERVER RESPONSE; GENERATE STATUS CODE IF TIMEOUT OR IF ERROR CODE RETURNED
150
164
HTML FILE RETURNED ?
YES
NO
166
PARSE HTML
RETURN

FIG. 10

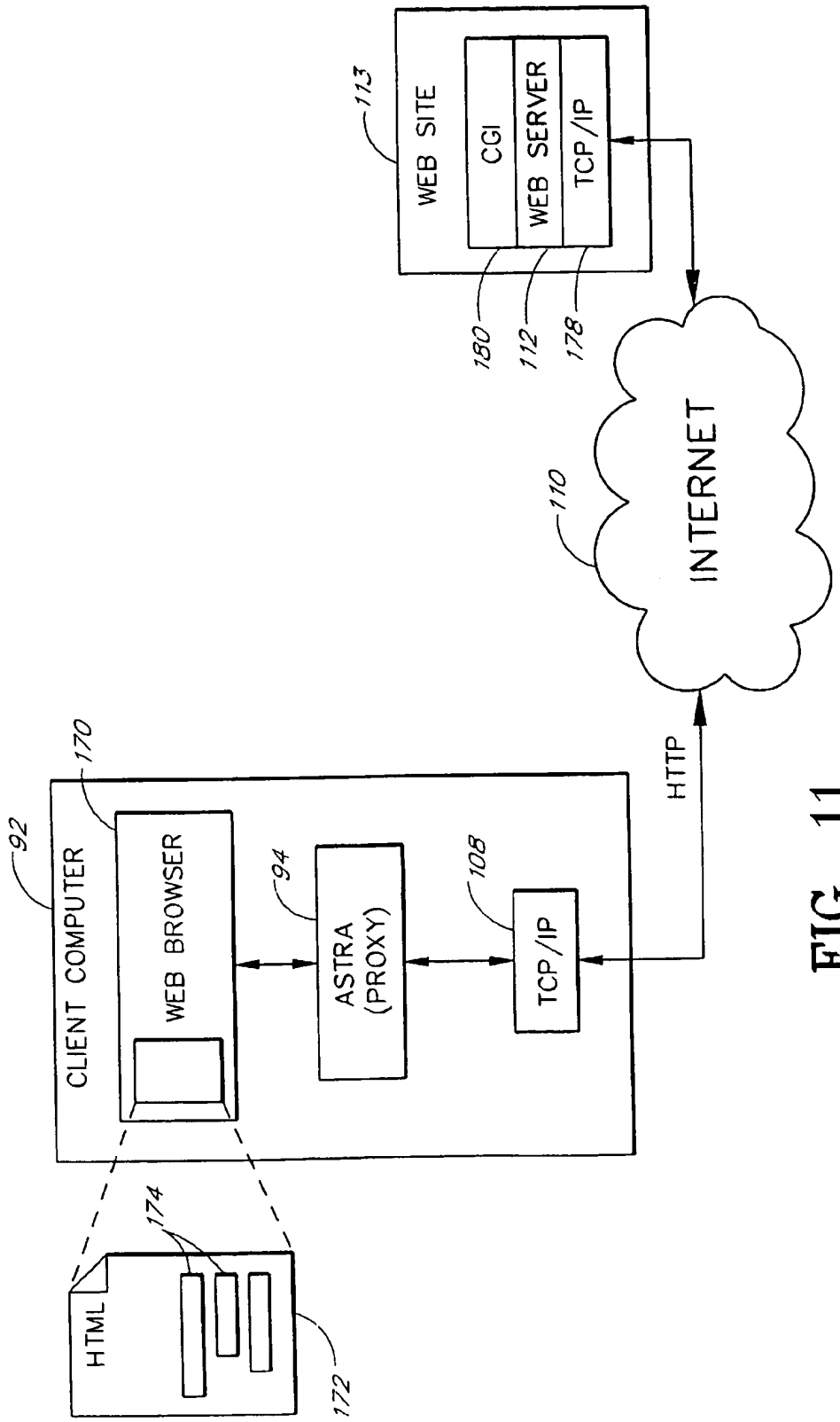

WEB SITE
113
CGI
WEB SERVER
TCP/IP
180
112
178
INTERNET
110
HTTP
CLIENT COMPUTER
92
170
WEB BROWSER
94
ASTRA (PROXY)
108
TCP/IP
HTML
174
172

FIG. 11

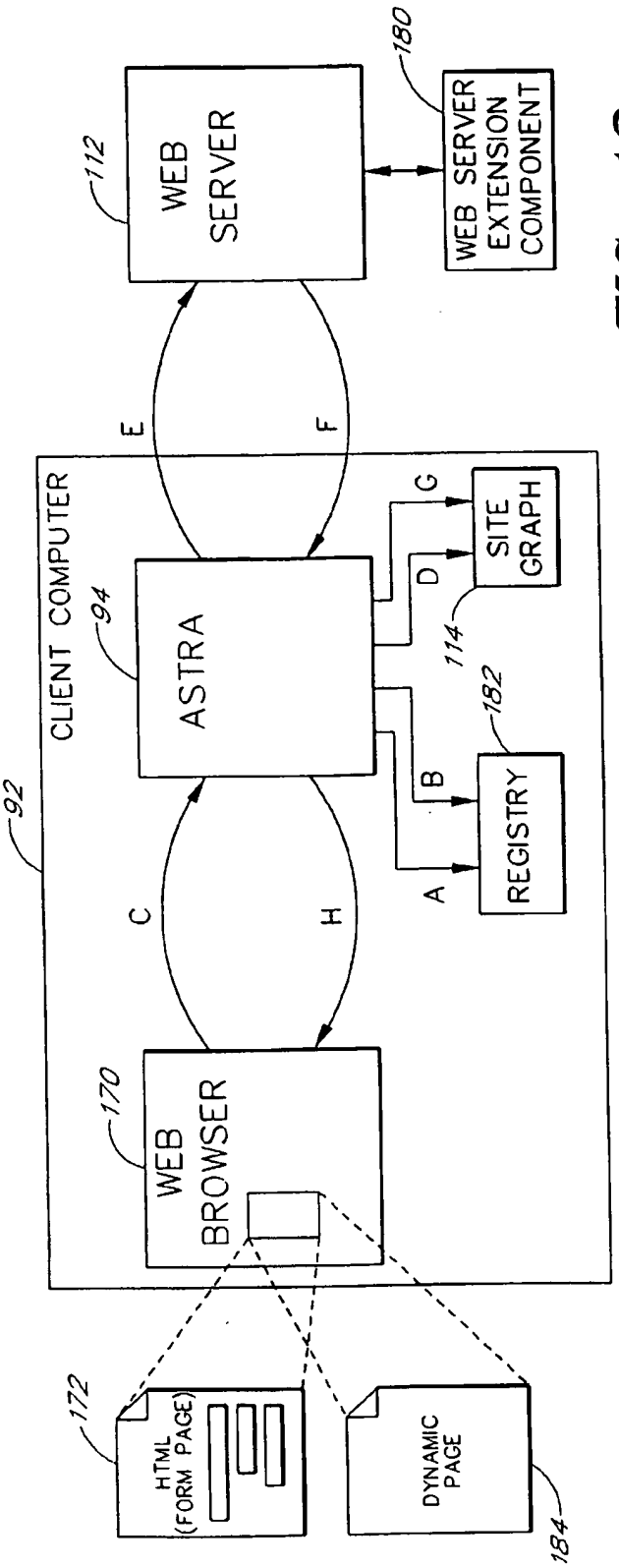


FIG. 12

A. ASTRA MODIFIES BROWSER CONFIGURATION IN REGISTRY TO SET ASTRA AS PROXY
B. ASTRA LAUNCHES BROWSER, AND THEN RESTORES ORIGINAL BROWSER CONFIGURATION WITHIN REGISTRY
C. BROWSER PASSES HTTP MESSAGE TO ASTRA IN RESPONSE TO SUBMISSION OF FORM
D. ASTRA EXTRACTS DATA SET AND STORES IN SITE GRAPH FOR FUTURE USE
E. ASTRA FORWARDS HTTP MESSAGE TO WEB SERVER
F. WEB SERVER RETURNS DYNAMICALLY-GENERATED WEB PAGE
G. WEB SERVER PARSES DYNAMICALLY-GENERATED PAGE AND UPDATES SITE GRAPH
H. ASTRA FORWARDS PAGE TO BROWSER TO CREATE IMPRESSION OF REGULAR BROWSING

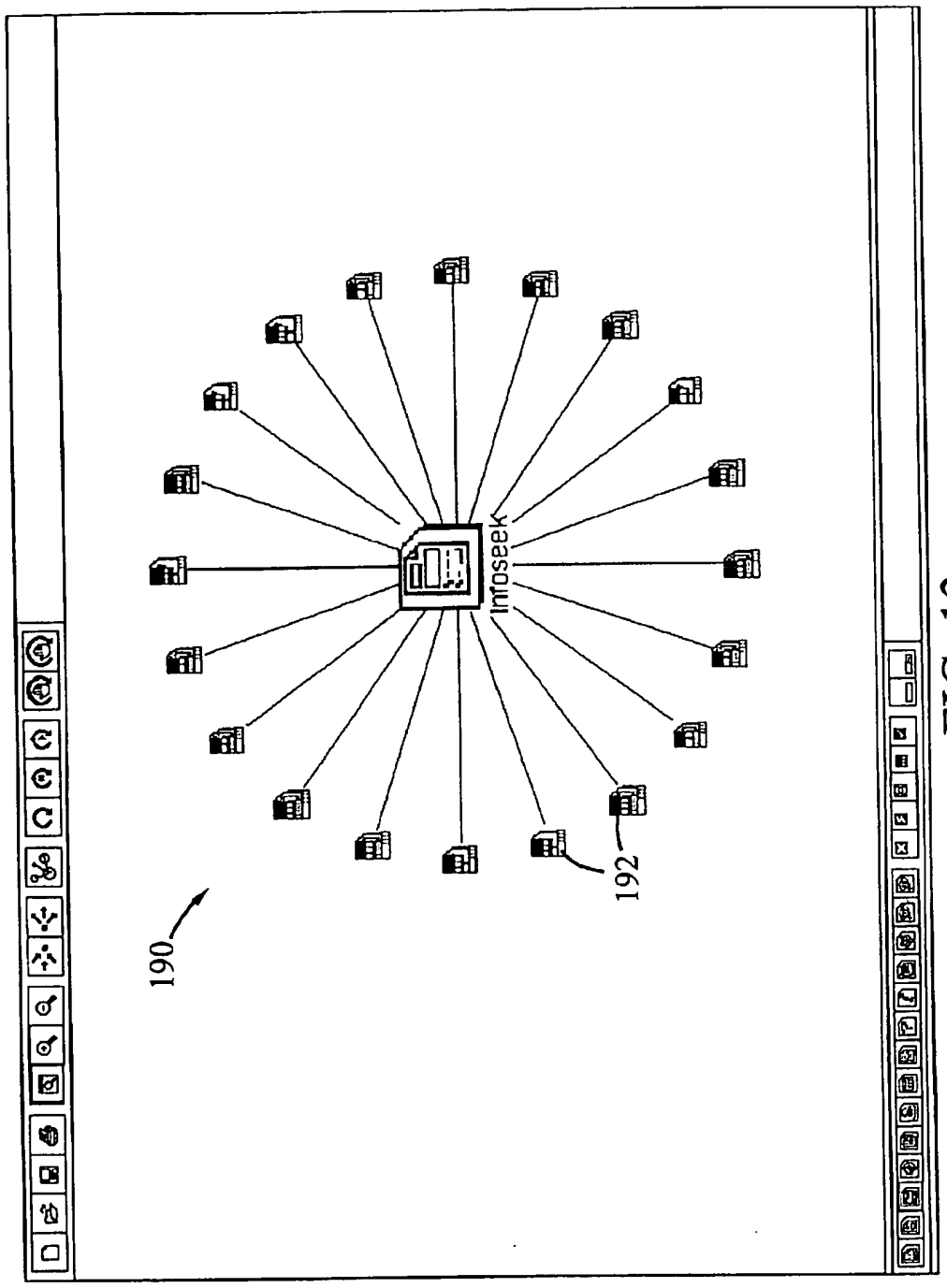


FIG. 13

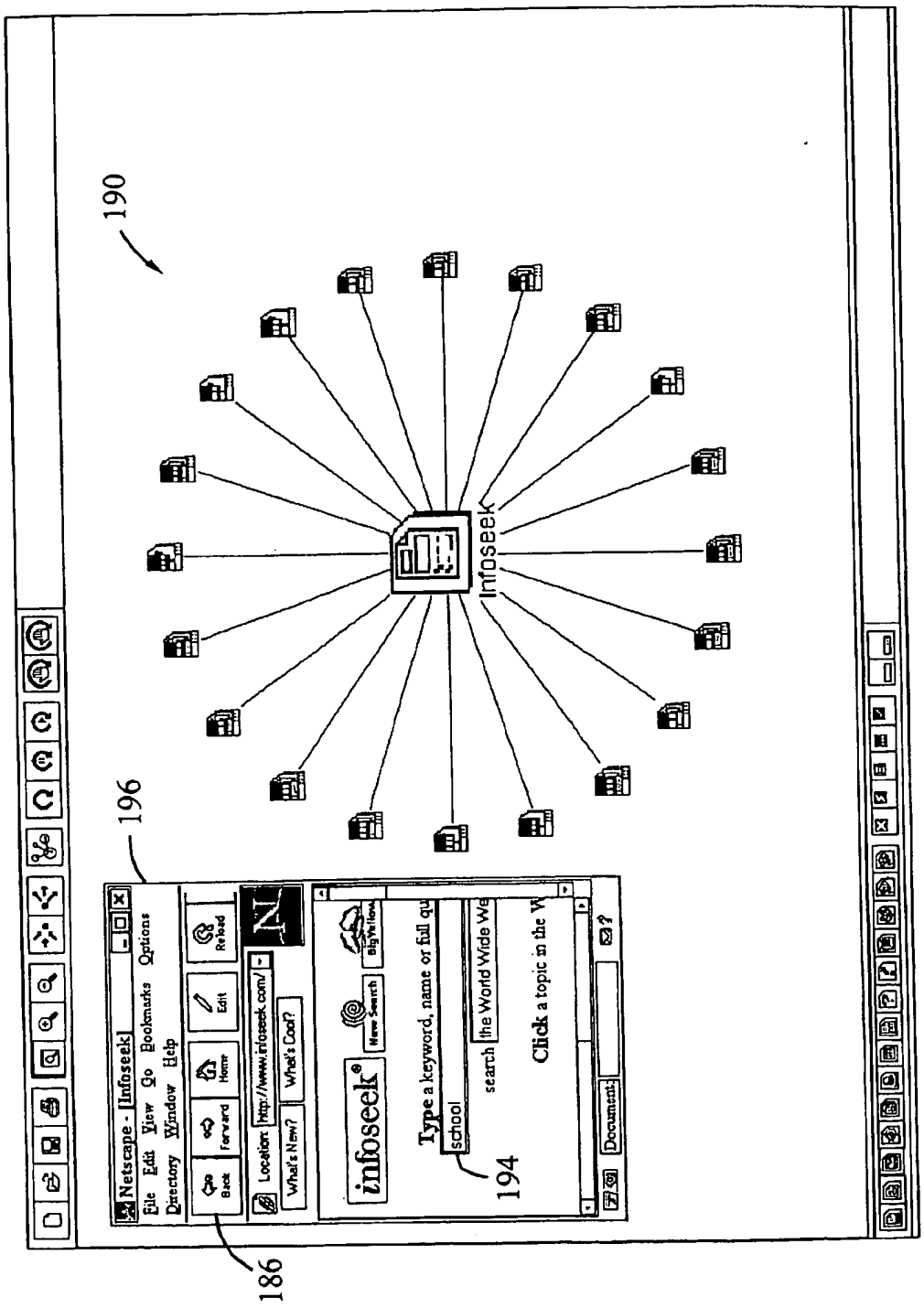

190
Infoseek
196
Netscape - [Infoseek]
File Edit View Go Bookmarks Options
Directory Window Help
Back
Forward
Home
Edit
Reload
Location: http://www.infoseek.com/
What's New?
What's Cool?
infoseek
Type a keyword, name or full qu
School
search
the World Wide We
Click a topic in the W
Documents
194
186

FIG. 14

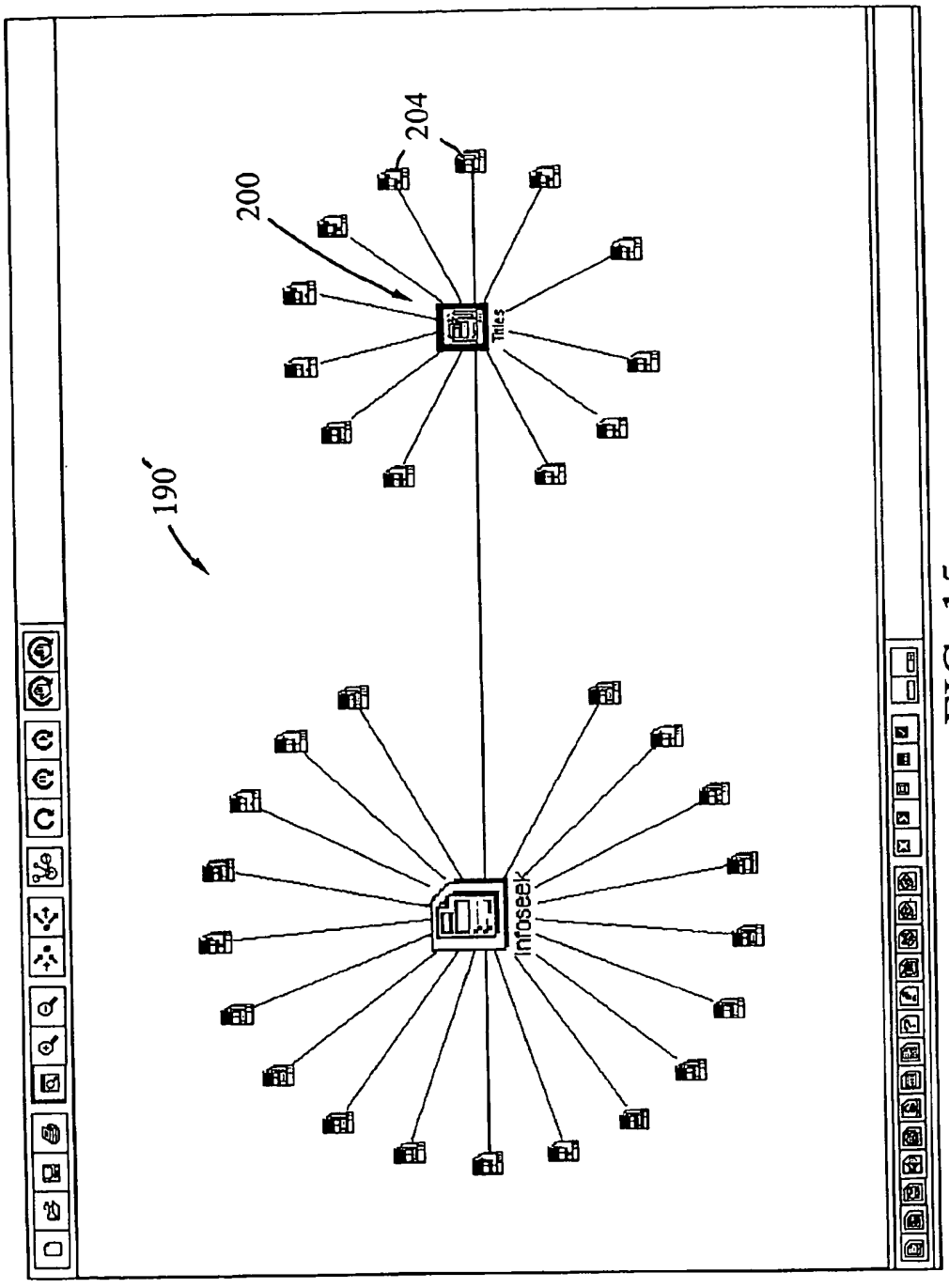


FIG. 15

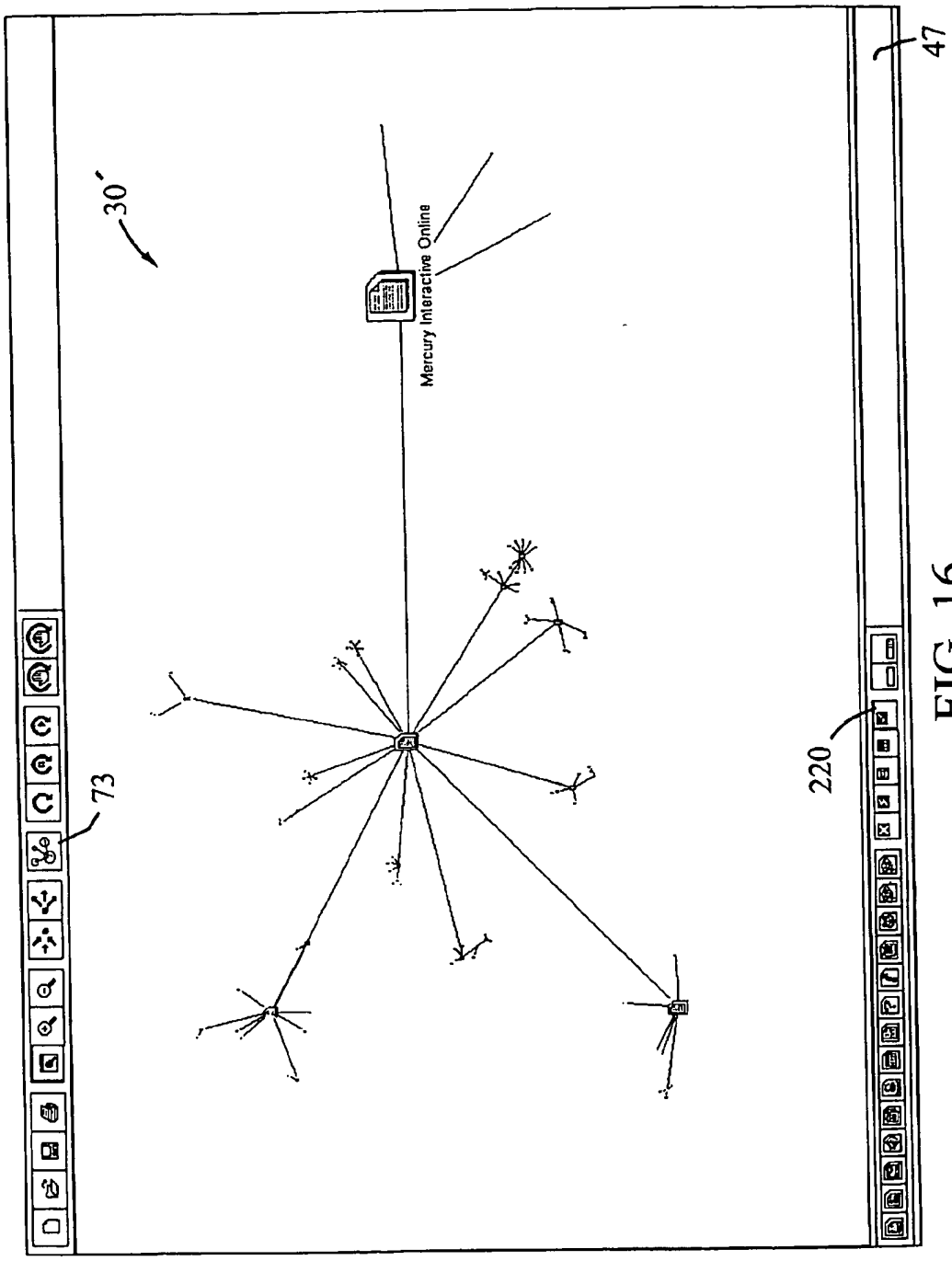


FIG. 16

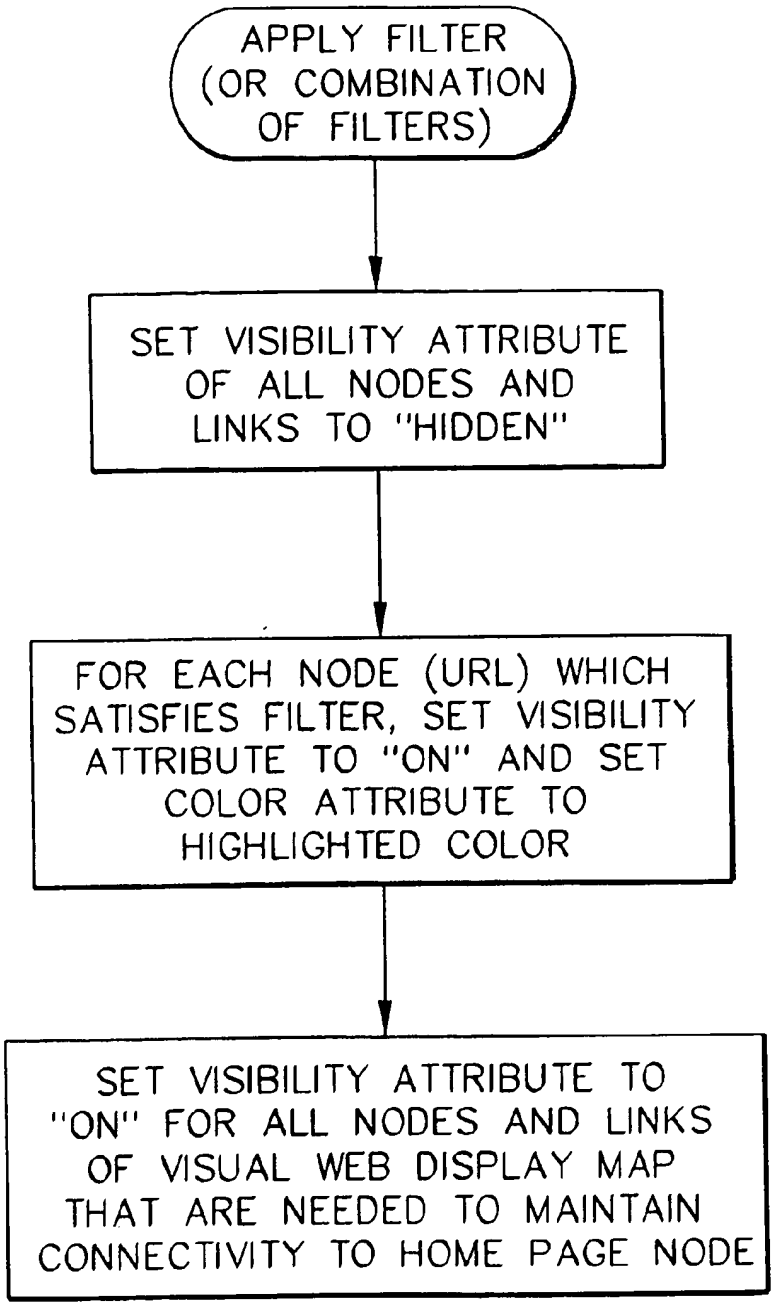

APPLY FILTER (OR COMBINATION OF FILTERS)
SET VISIBILITY ATTRIBUTE OF ALL NODES AND LINKS TO "HIDDEN"
FOR EACH NODE (URL) WHICH SATISFIES FILTER, SET VISIBILITY ATTRIBUTE TO "ON" AND SET COLOR ATTRIBUTE TO HIGHLIGHTED COLOR
SET VISIBILITY ATTRIBUTE TO "ON" FOR ALL NODES AND LINKS OF VISUAL WEB DISPLAY MAP THAT ARE NEEDED TO MAINTAIN CONNECTIVITY TO HOME PAGE NODE

FIG. 17

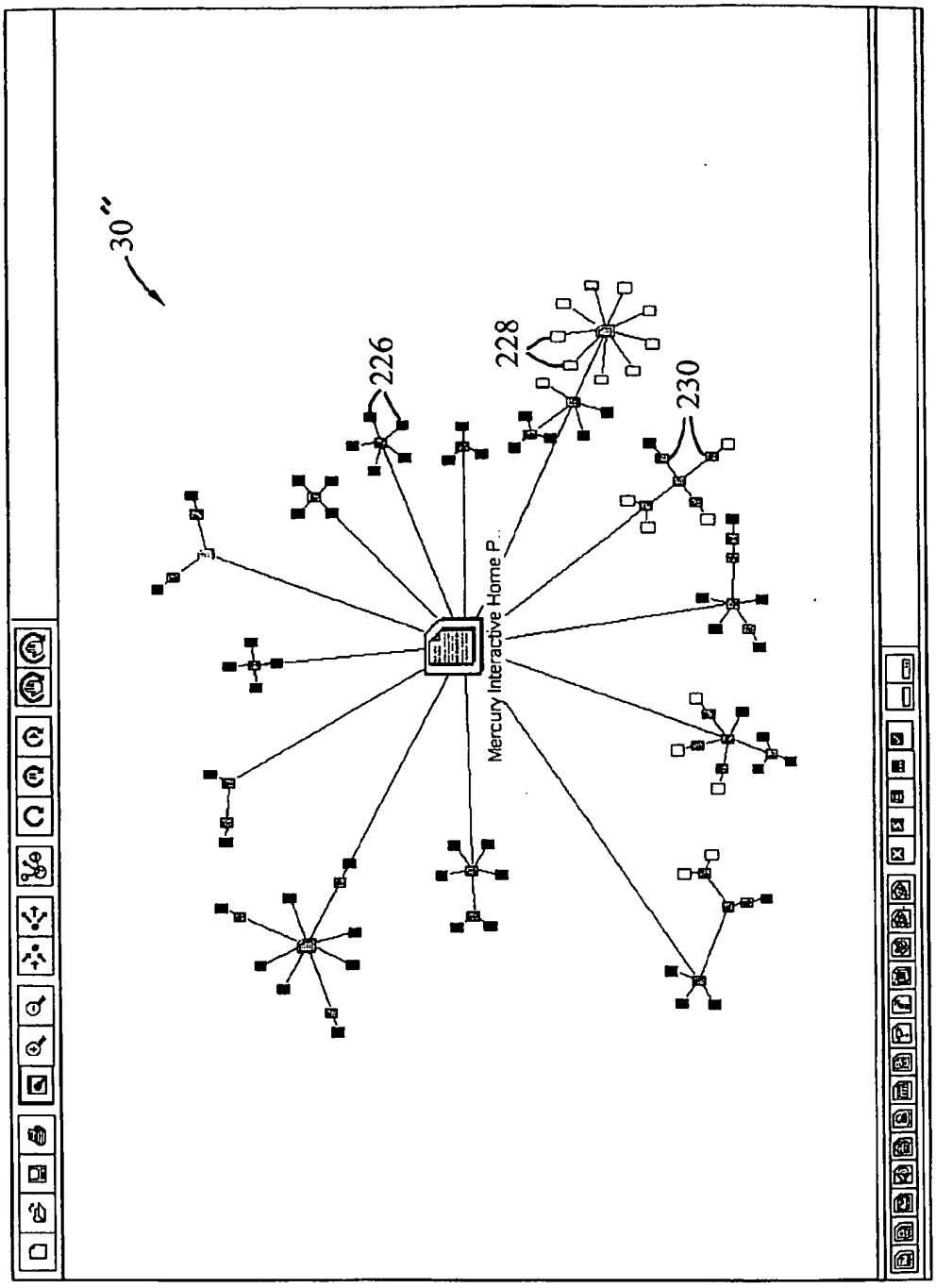


FIG. 18

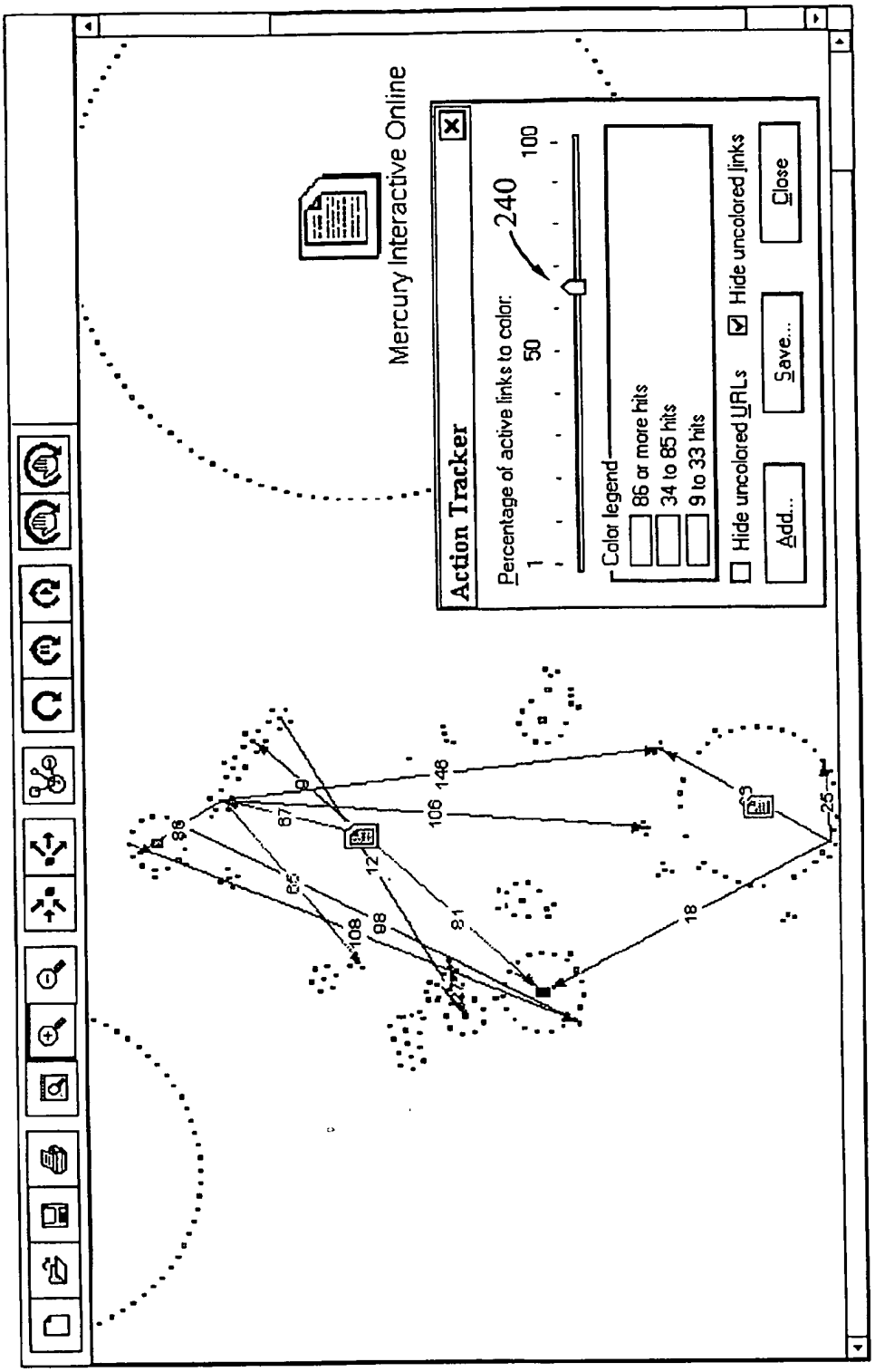

Mercury Interactive Online
Action Tracker
Percentage of active links to color:
1
50
100
240
Color legend
86 or more hits
34 to 85 hits
9 to 33 hits
Hide uncolored URLs
Hide uncolored links
Add...
Save...
Close
148
106
87
86
12
65
108
98
81
18
25

FIG. 19

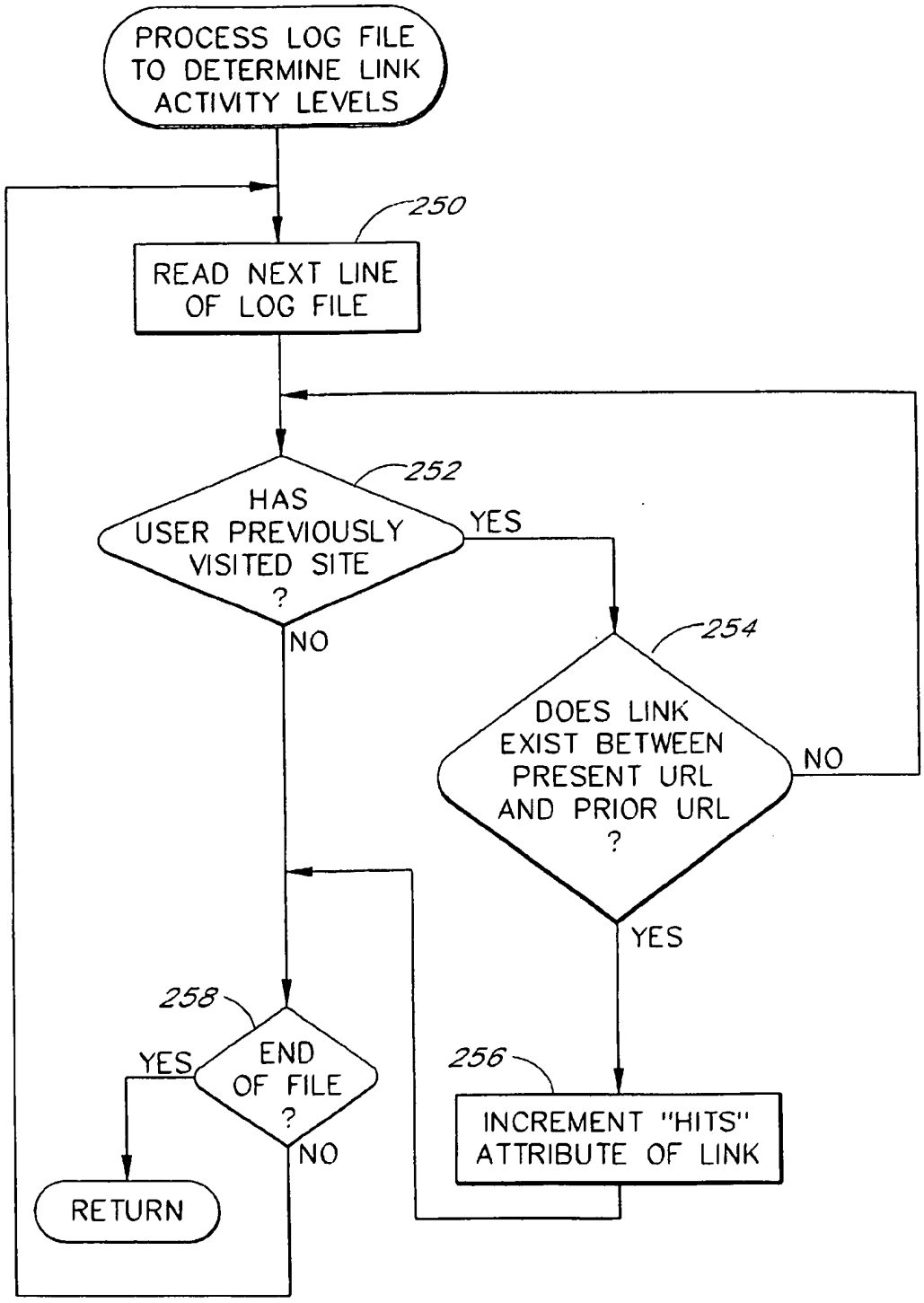

PROCESS LOG FILE TO DETERMINE LINK ACTIVITY LEVELS
250
READ NEXT LINE OF LOG FILE
252
HAS USER PREVIOUSLY VISITED SITE ?
YES
NO
254
DOES LINK EXIST BETWEEN PRESENT URL AND PRIOR URL ?
NO
YES
256
INCREMENT "HITS" ATTRIBUTE OF LINK
258
END OF FILE ?
YES
NO
RETURN

FIG. 20

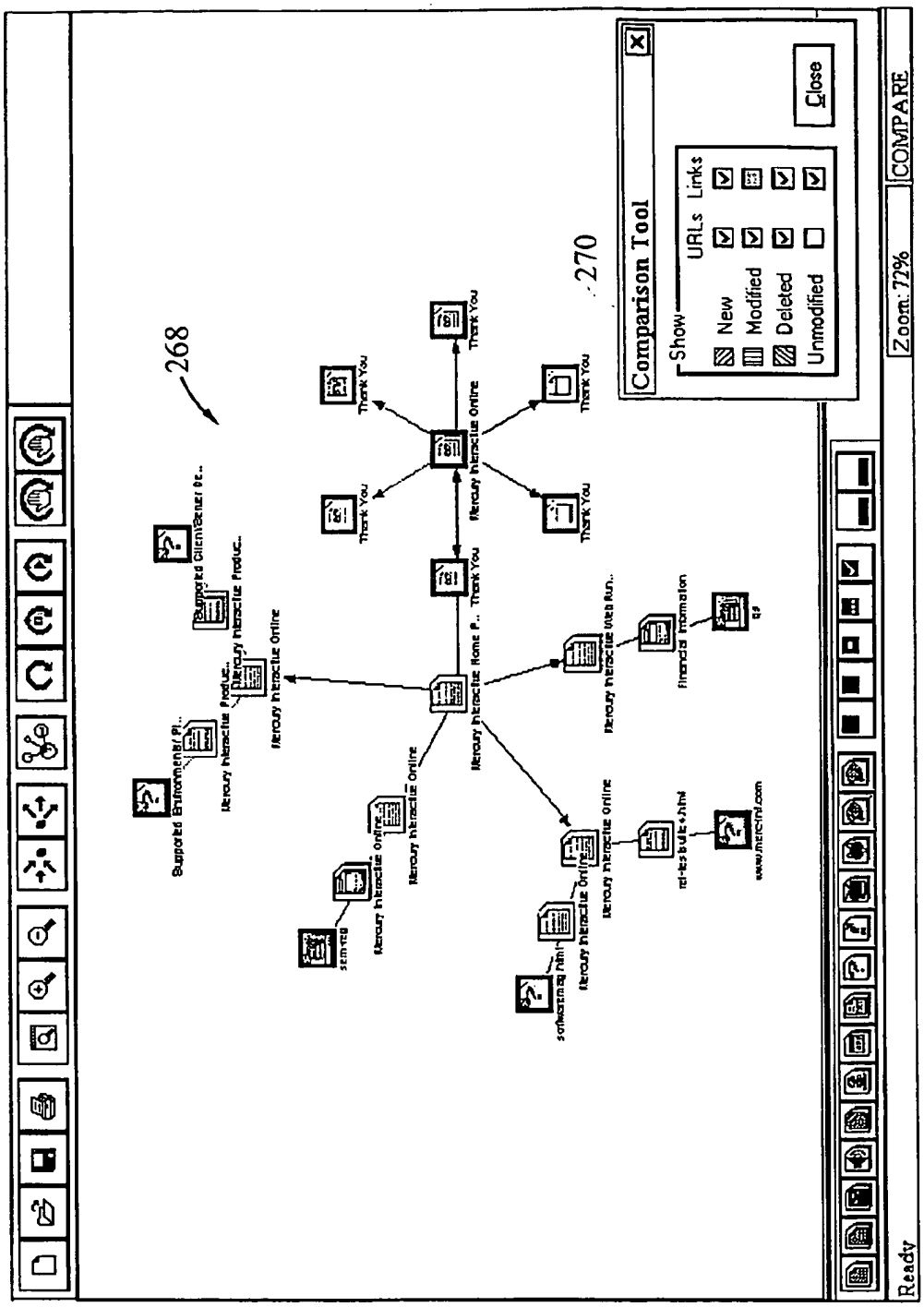

268
270
Comparison Tool
Show
URLs
Links
New
Modified
Deleted
Unmodified
Close
Zoom: 72%
COMPARE
Ready

FIG. 21

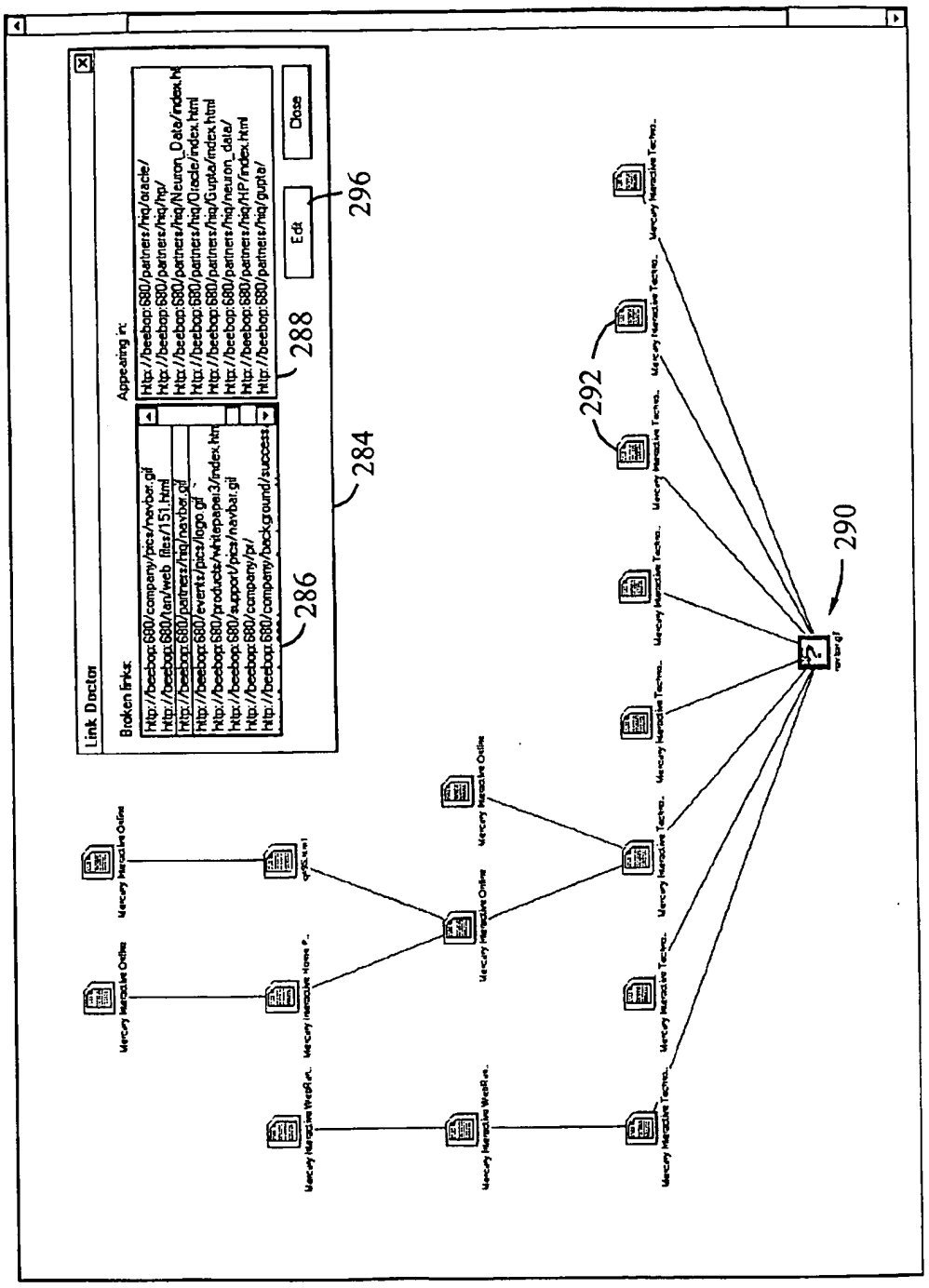

Link Doctor
Broken links:
Appearing in:
Edit
Done
284
286
288
290
292
296

FIG. 22

FIG. 23

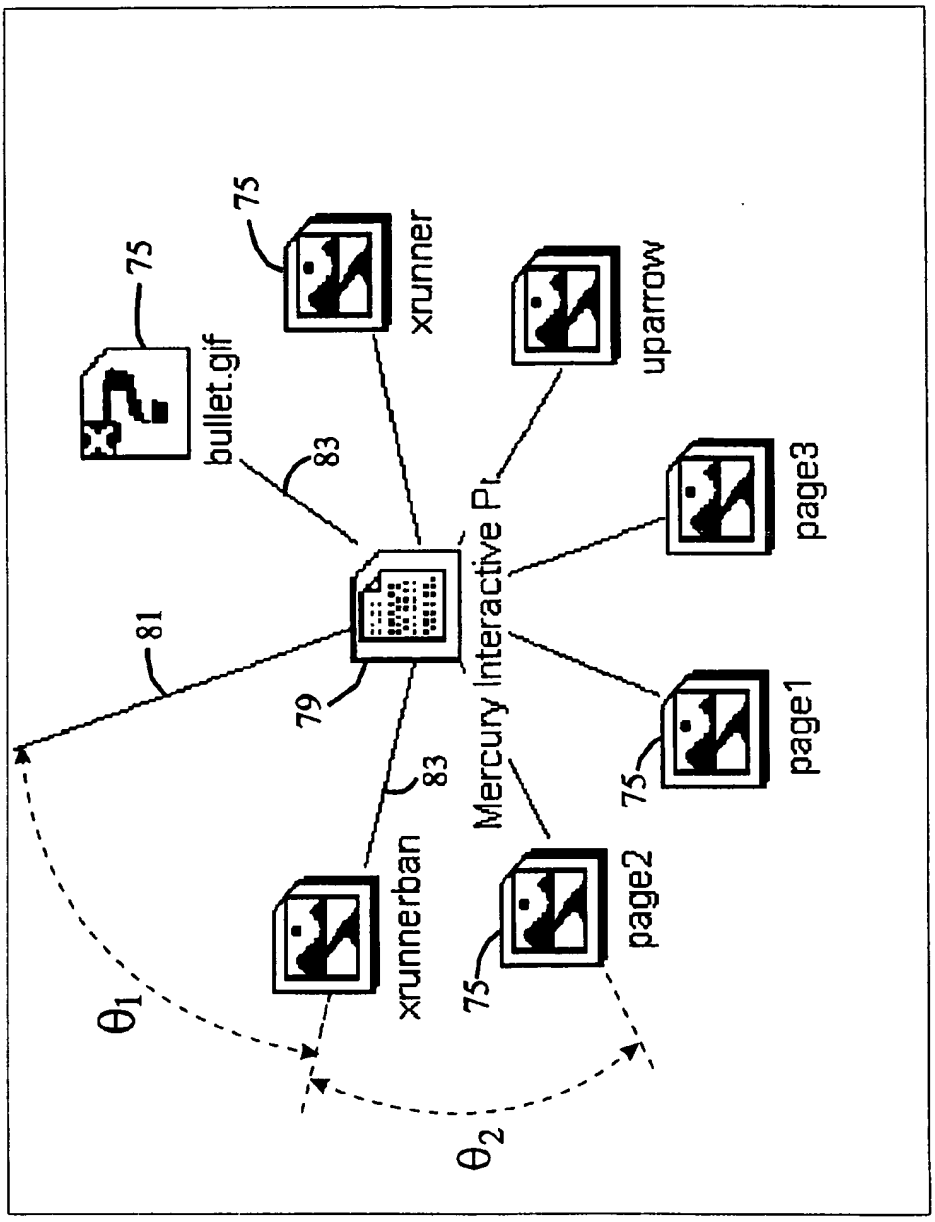
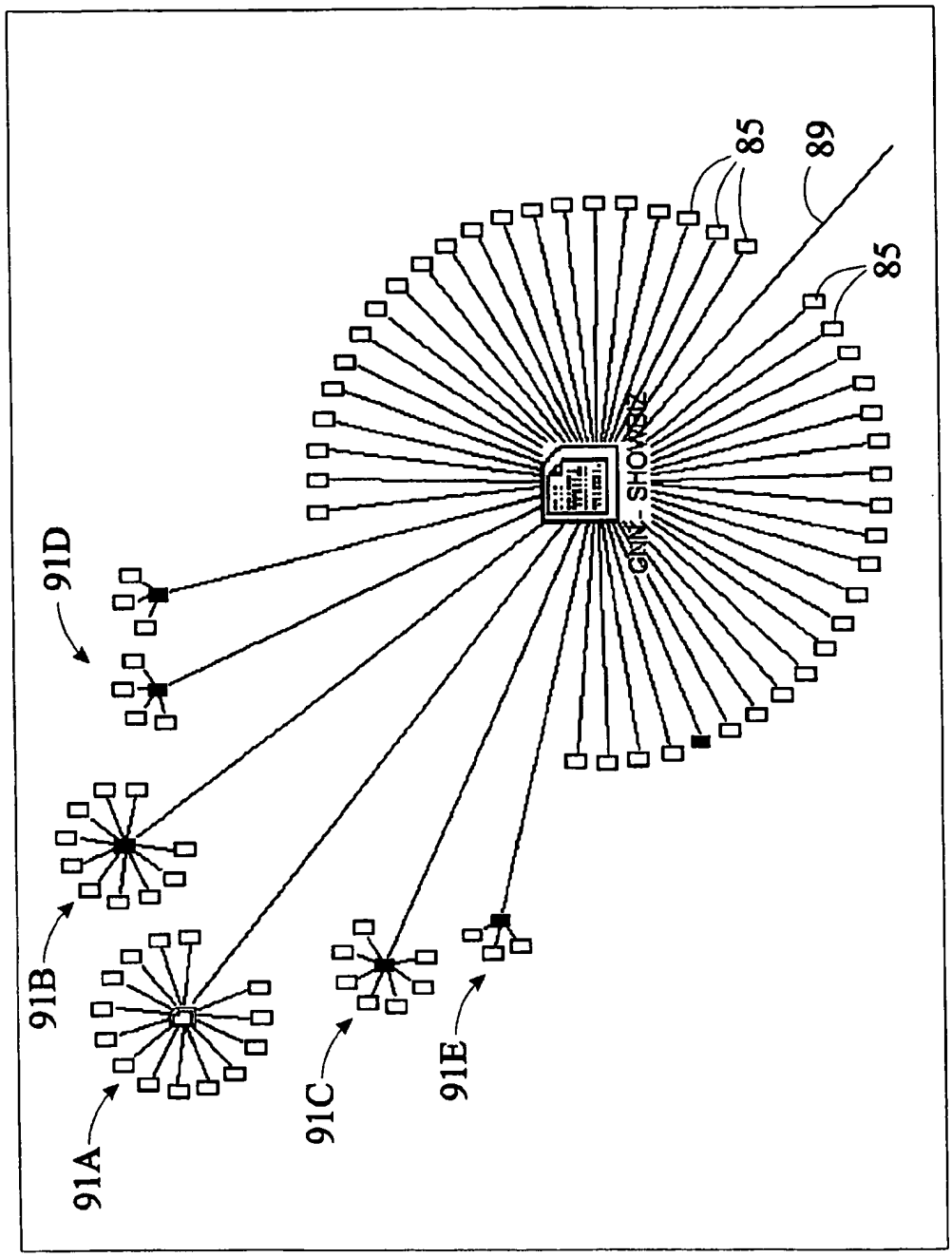

75
bullet.gif
75
xrunner
uparrow
83
page3
Mercury Interactive Pt
81
79
page1
75
83
xrunnerban
page2
75
θ1
θ2

FIG. 24



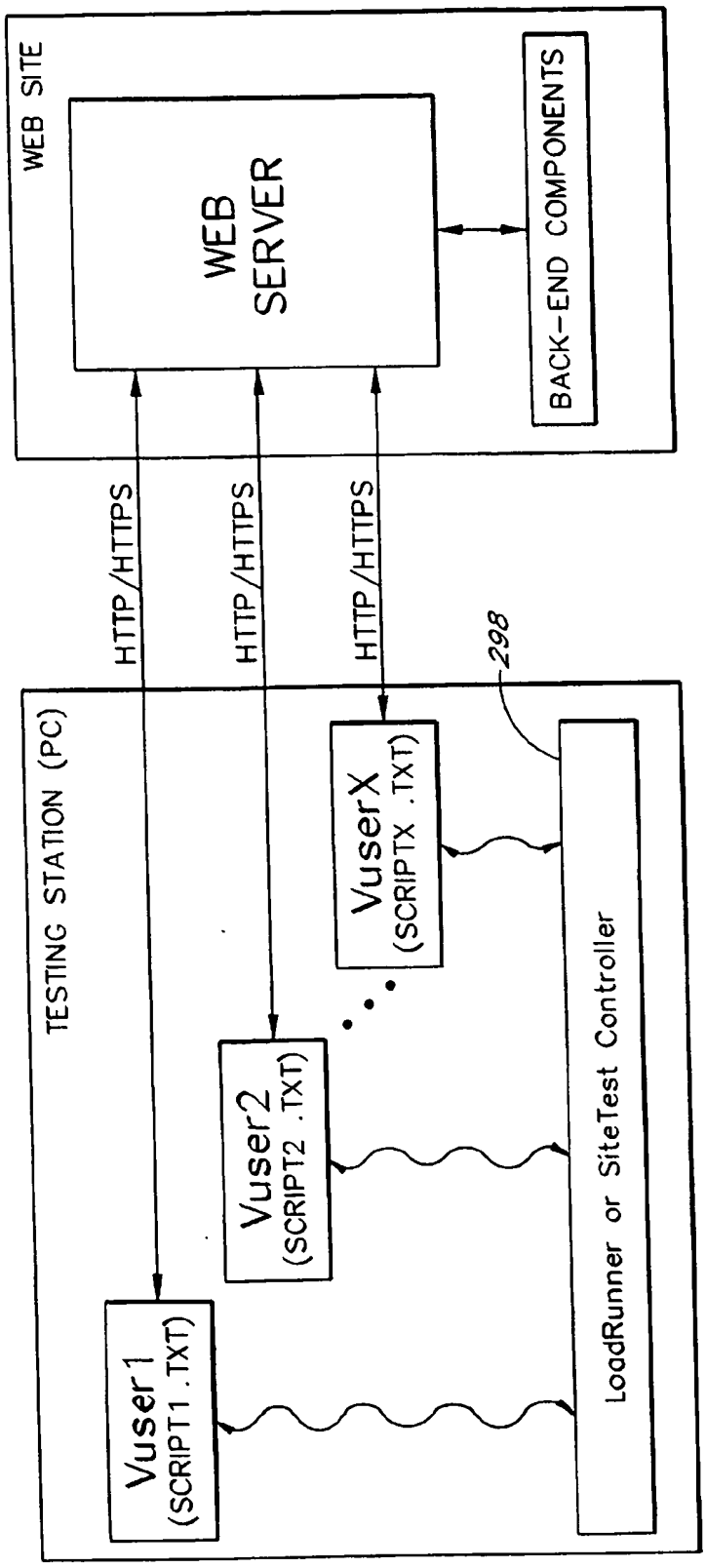

WEB SITE
WEB SERVER
BACK-END COMPONENTS
HTTP/HTTPS
HTTP/HTTPS
HTTP/HTTPS
298
TESTING STATION (PC)
VuserX (SCRIPTX .TXT)
Vuser2 (SCRIPT2 .TXT)
Vuser1 (SCRIPT1 .TXT)
LoadRunner or SiteTest Controller

FIG. 25

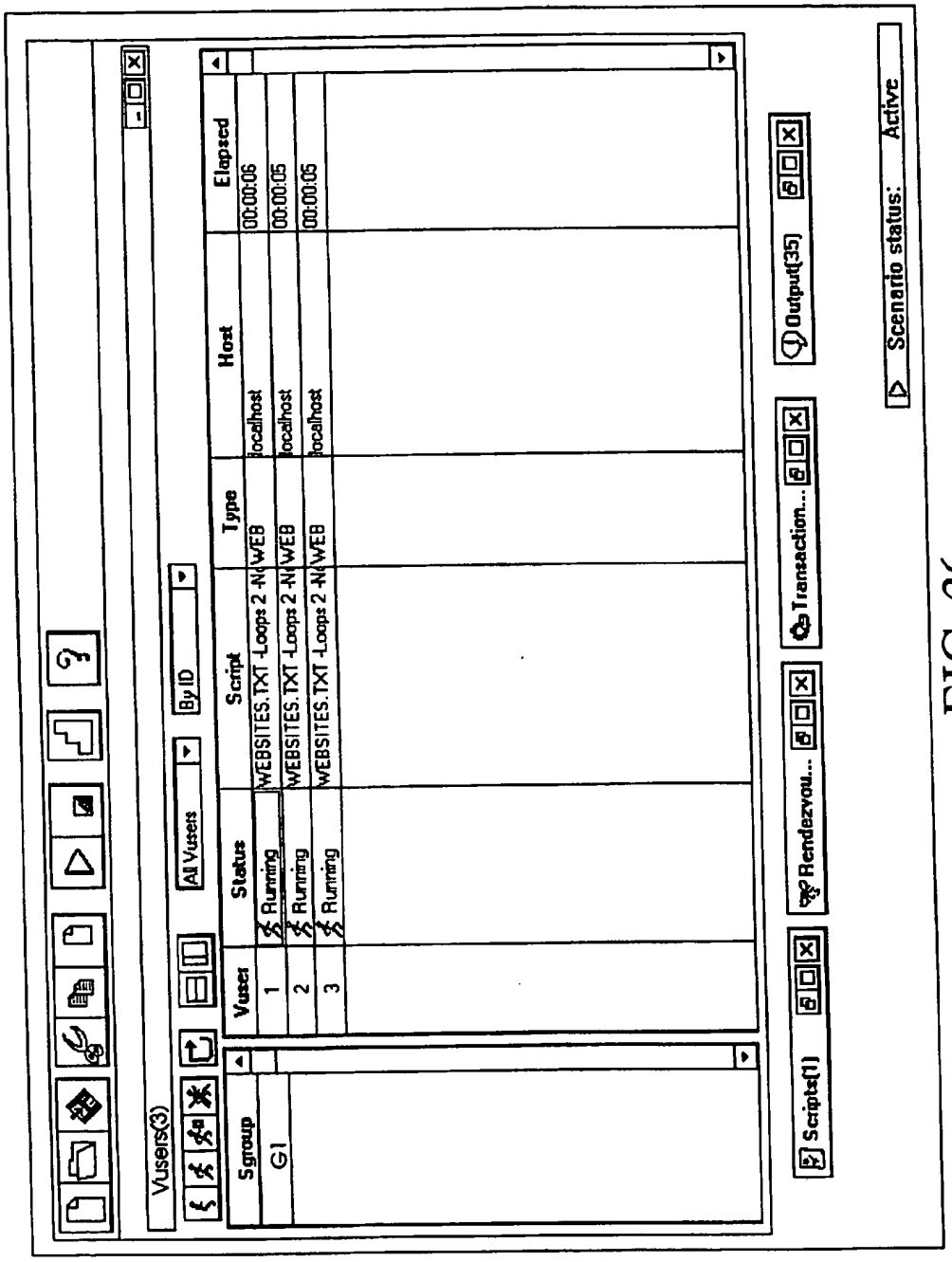

Vusers(3)
All Vusers
By ID
Sgroup
G1
Vuser
Status
Script
Type
Host
Elapsed
1
Running
WEBSITES.TXT -Loops 2 -N
WEB
localhost
00:00:06
2
Running
WEBSITES.TXT -Loops 2 -N
WEB
localhost
00:00:05
3
Running
WEBSITES.TXT -Loops 2 -N
WEB
localhost
00:00:05
Scripts(1)
Rendezvou...
Transaction...
Output(35)
Scenario status: Active

FIG. 26

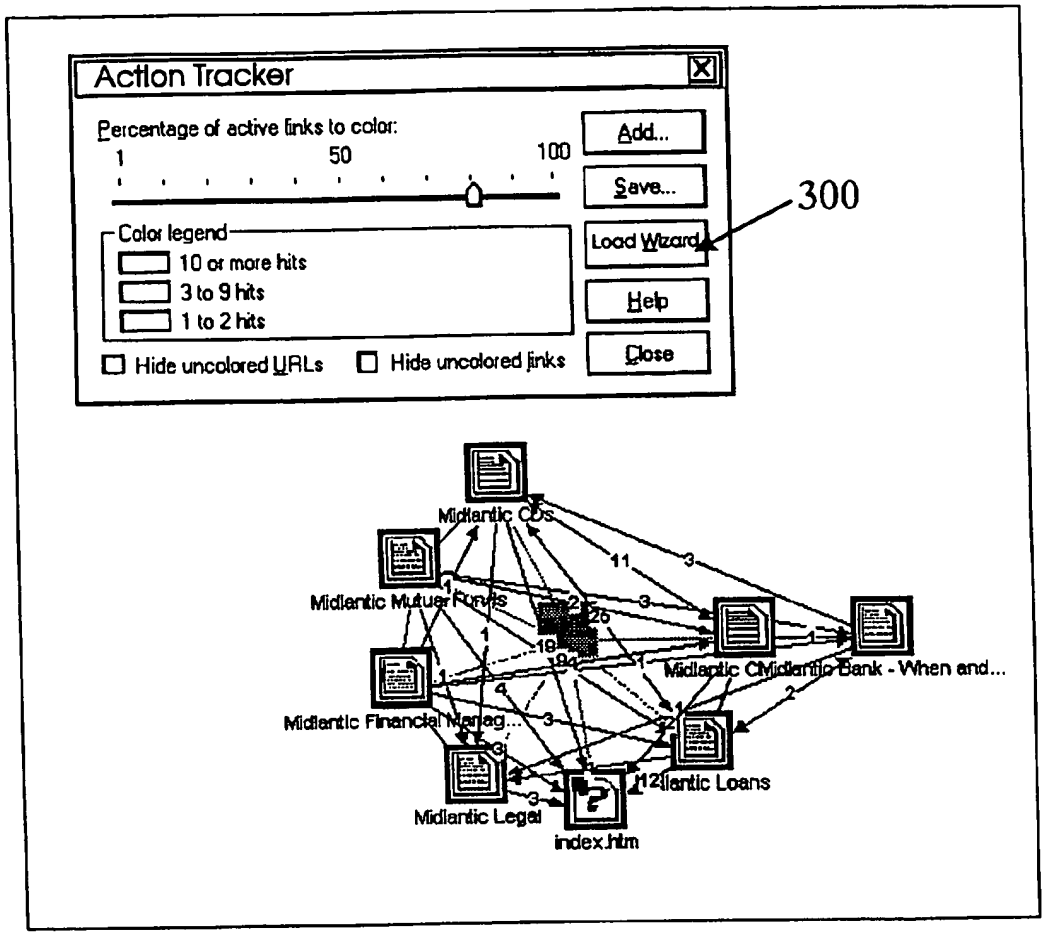

Action Tracker
Percentage of active links to color:
1
50
100
Add...
Save...
Load Wizard
Help
Close
300
Color legend
10 or more hits
3 to 9 hits
1 to 2 hits
Hide uncolored URLs
Hide uncolored links
Midlantic COs
Midlantic Mutuel Funds
Midlantic Financial Manag...
Midlantic Legal
index.htm
Midlantic Loans
Midlantic Bank - When and...

FIG. 27

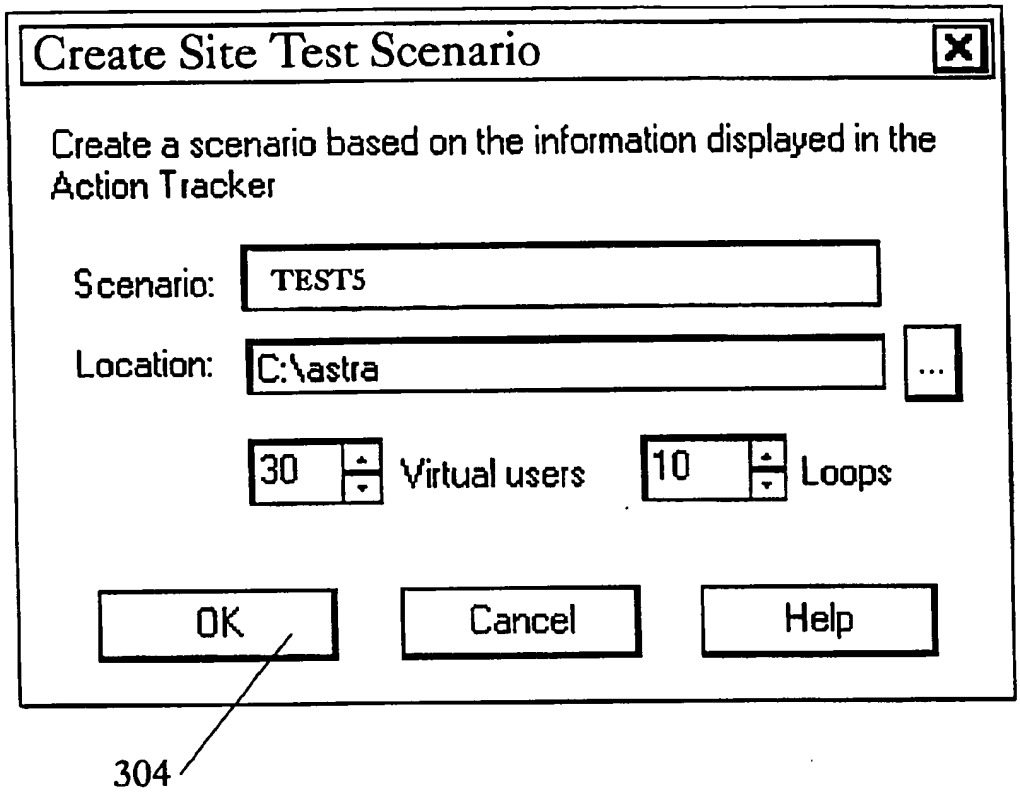

Create Site Test Scenario
Create a scenario based on the information displayed in the Action Tracker
Scenario:
TEST5
Location:
C:\astra
...
30
Virtual users
10
Loops
OK
Cancel
Help
304

FIG. 28

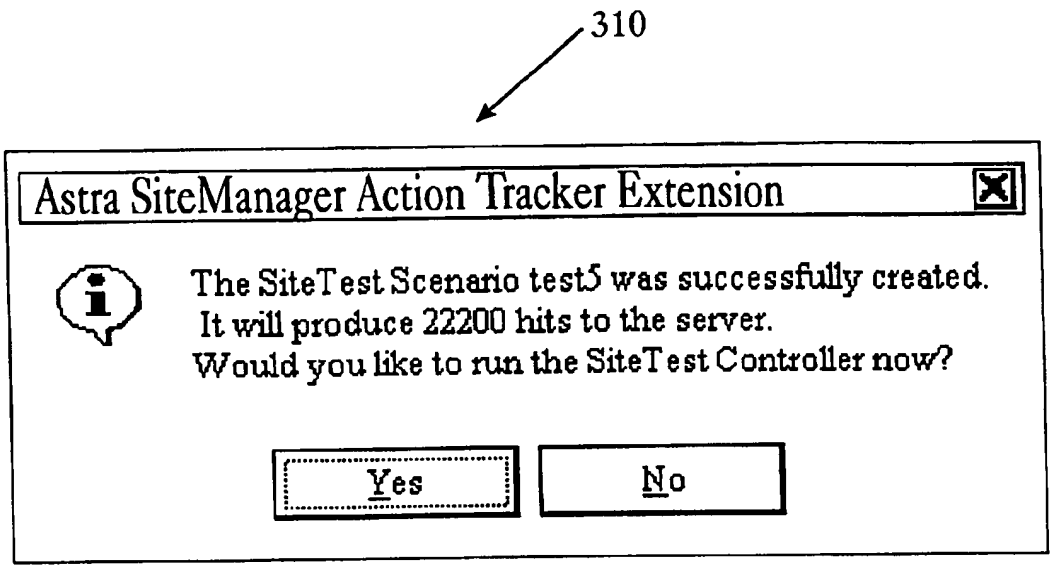

310
Astra SiteManager Action Tracker Extension
The SiteTest Scenario test5 was successfully created.
It will produce 22200 hits to the server.
Would you like to run the SiteTest Controller now?
Yes
No

FIG. 29

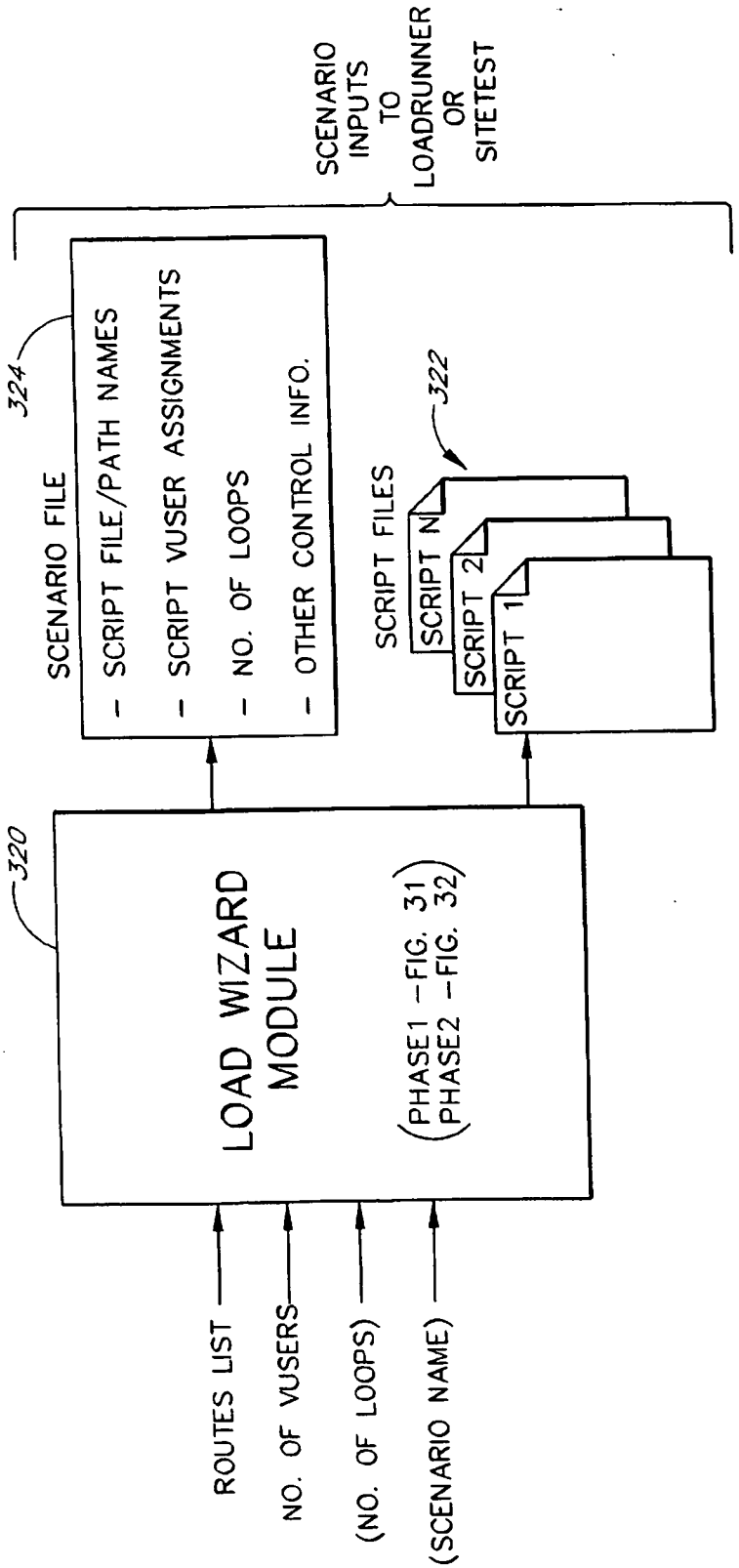

SCENARIO INPUTS TO LOADRUNNER OR SITETEST
324
SCENARIO FILE
- SCRIPT FILE/PATH NAMES
- SCRIPT VUSER ASSIGNMENTS
- NO. OF LOOPS
- OTHER CONTROL INFO.
322
SCRIPT FILES
SCRIPT N
SCRIPT 2
SCRIPT 1
320
LOAD WIZARD MODULE
(PHASE1 -FIG. 31
PHASE2 -FIG. 32)
ROUTES LIST
NO. OF VUSERS
(NO. OF LOOPS)
(SCENARIO NAME)

FIG. 30

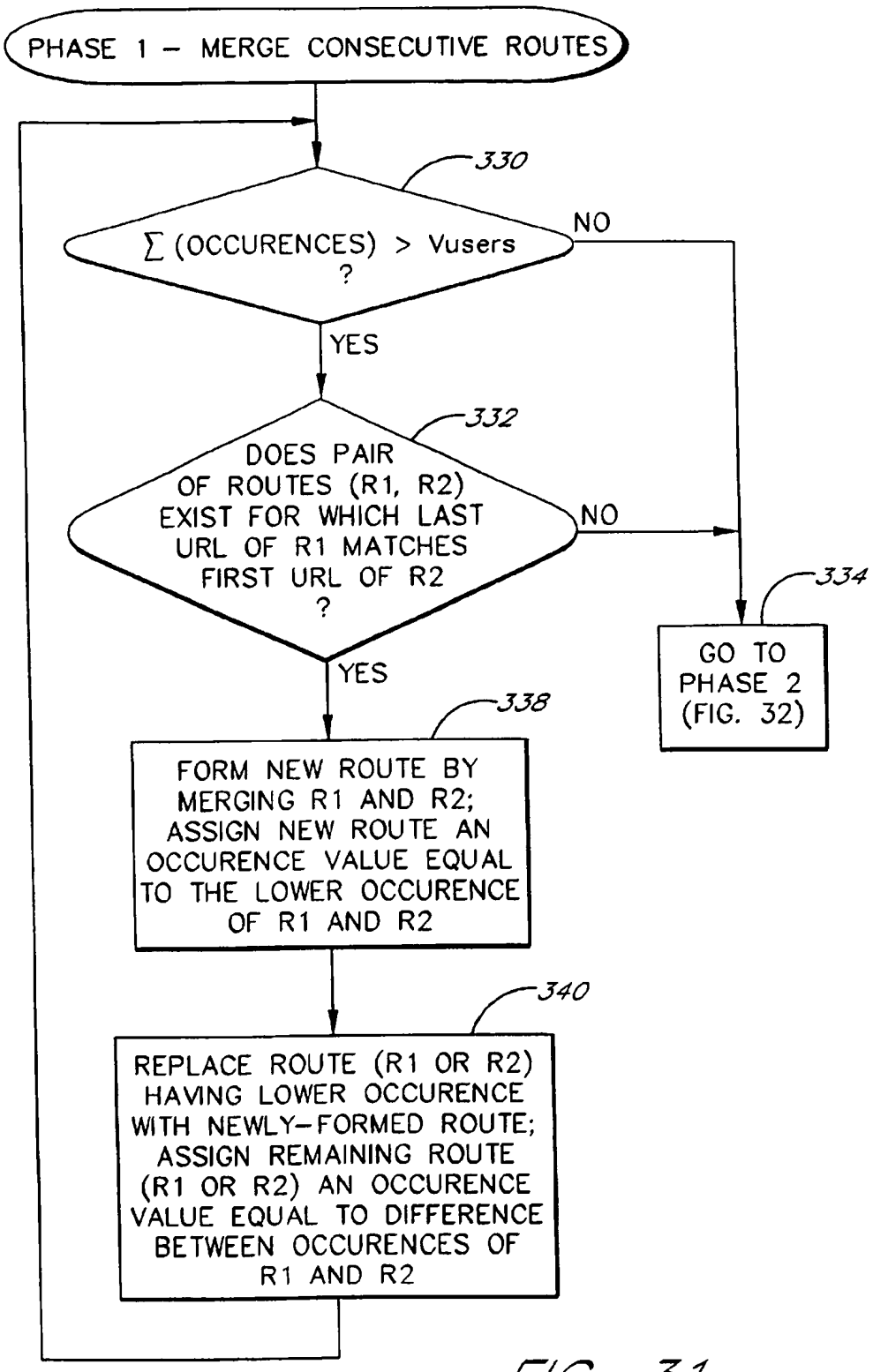

PHASE 1 – MERGE CONSECUTIVE ROUTES
330
Σ (OCCURENCES) > Vusers ?
NO
YES
332
DOES PAIR OF ROUTES (R1, R2) EXIST FOR WHICH LAST URL OF R1 MATCHES FIRST URL OF R2 ?
NO
334
GO TO PHASE 2 (FIG. 32)
YES
338
FORM NEW ROUTE BY MERGING R1 AND R2; ASSIGN NEW ROUTE AN OCCURENCE VALUE EQUAL TO THE LOWER OCCURENCE OF R1 AND R2
340
REPLACE ROUTE (R1 OR R2) HAVING LOWER OCCURENCE WITH NEWLY-FORMED ROUTE; ASSIGN REMAINING ROUTE (R1 OR R2) AN OCCURENCE VALUE EQUAL TO DIFFERENCE BETWEEN OCCURENCES OF R1 AND R2

FIG. 31

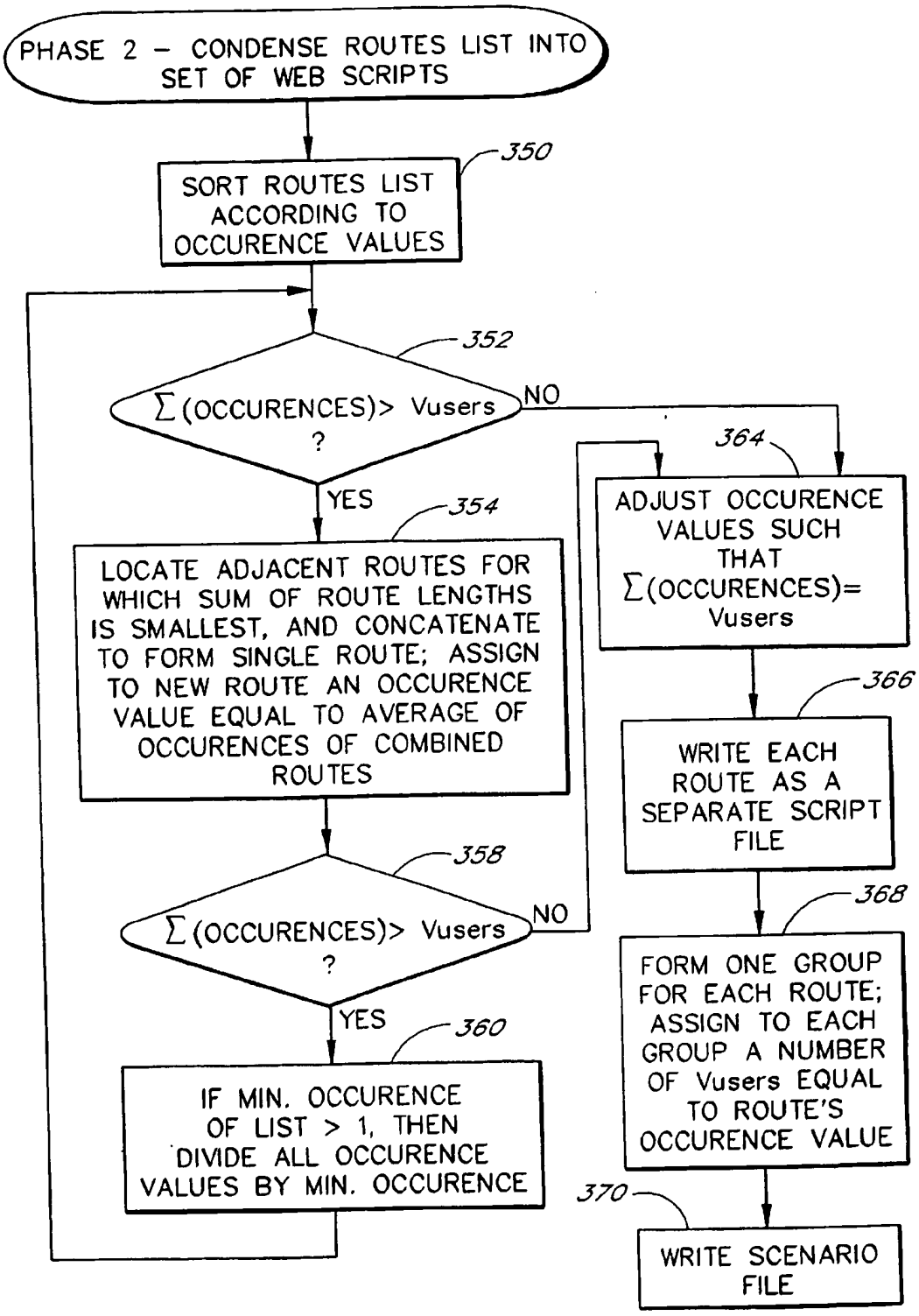

PHASE 2 – CONDENSE ROUTES LIST INTO SET OF WEB SCRIPTS
350
SORT ROUTES LIST ACCORDING TO OCCURENCE VALUES
352
Σ(OCCURENCES)> Vusers ?
NO
YES
354
LOCATE ADJACENT ROUTES FOR WHICH SUM OF ROUTE LENGTHS IS SMALLEST, AND CONCATENATE TO FORM SINGLE ROUTE; ASSIGN TO NEW ROUTE AN OCCURENCE VALUE EQUAL TO AVERAGE OF OCCURENCES OF COMBINED ROUTES
358
Σ(OCCURENCES)> Vusers ?
NO
YES
360
IF MIN. OCCURENCE OF LIST > 1, THEN DIVIDE ALL OCCURENCE VALUES BY MIN. OCCURENCE
364
ADJUST OCCURENCE VALUES SUCH THAT Σ(OCCURENCES)= Vusers
366
WRITE EACH ROUTE AS A SEPARATE SCRIPT FILE
368
FORM ONE GROUP FOR EACH ROUTE; ASSIGN TO EACH GROUP A NUMBER OF Vusers EQUAL TO ROUTE'S OCCURENCE VALUE
370
WRITE SCENARIO FILE

FIG. 32

USE OF SERVER ACCESS LOGS TO GENERATE SCRIPTS AND SCENARIOS FOR EXERCISING AND EVALUATING PERFORMANCE OF WEB SITES

PRIORITY CLAIM

This application is a continuation of application Ser. No. 09/315,795, filed May 21, 1999, which is a continuation of Appl. Ser. No. 08/949,680, filed Oct. 14, 1997 (now U.S. Pat. No. 5,974,572), which is a continuation-in-part of application Ser. No. 08/840,103, filed Apr. 11, 1997, which claims the benefit of U.S. Provisional Appl. No. 60/028,474, filed Oct. 15, 1996.

MICROFICHE APPENDIX

This specification includes a microfiche appendix (consisting of 1 sheet with 51 frames) which contains partial source code listings (Appendices A and C) and an application program interface specification (Appendix B) of a software product that embodies the invention. These materials form part of the disclosure of the specification.

FIELD OF THE INVENTION

The present invention relates to software tools for load-testing Web sites and other types of client-server systems. More particularly, the invention relates to a method of efficiently generating a load testing scenario that allows a Web site to be tested according to browsing behaviors of typical users.

BACKGROUND OF THE INVENTION

With the increasing popularity and complexity of Internet and intranet applications, the task of managing Web site content and maintaining Web site effectiveness has become increasingly difficult. Company Webmasters and business managers are routinely faced with a wide array of burdensome tasks, including, for example, the identification and repair of large numbers of broken links (i.e., links to missing URLs), the monitoring and organization of large volumes of diverse, continuously-changing Web site content, and the detection and management of congested links. These problems are particularly troublesome for companies that rely on their respective Web sites to provide mission-critical information and services to customers and business partners.

Several software companies have developed software products which address some of these problems by generating graphical maps of Web site content and providing tools for navigating and managing the content displayed within the maps. Examples of such software tools include Web-Mapper™ from Netcarta Corporation and WebAnalyzer™ from InContext Corporation. Unfortunately, the graphical site maps generated by these products tend to be difficult to navigate, and fail to convey much of the information needed by Webmasters to effectively manage complex Web sites. As a result, many companies continue to resort to the burdensome task of manually generating large, paper-based maps of their Web sites. In addition, many of these products are only capable of mapping certain types of Web pages, and do not provide the types of analysis tools needed by Webmasters to evaluate the performance and effectiveness of Web sites.

Another problem in the field of Web site and intranet management relates to the ability of the site to handle peak loads. The heavy use of a site can lead to a significant degradation in performance, or even a complete loss of service. Examples of Web sites that have undergone severe performance degradations during heavy use include the IBM Olympic site during the 1996 summer olympics, and the CNN Interactive Election Day site during the 1996 presidential election.

Mercury Interactive Corporation, the assignee of the present application, has addressed this problem by developing software tools that allow companies to load-test their World Wide Web and intranet sites. Mercury Interactive's LoadRunner® 4.5 and Astra™ SiteTest 1.0 products, for example, include functionality for sending large volumes of client requests (representing hundreds or thousands of concurrent users) to a site while monitoring the site's performance. These products use a virtual user executable, or "Vuser," to send the client requests to the Web site while monitoring the site's performance. The Vuser sends the client requests to the site according to a pre-defined test script (also referred to as a "Vuser script" or "Web script"), which is in the general form of a list of the HTTP (HyperText Transport Protocol) messages to be sent to the site. In one implementation, up to 50 Vusers can be run concurrently on a single Windows® or 95 workstation, with different Vusers using different test scripts.

SUMMARY OF THE INVENTION

To facilitate the generation of the test scripts, the LoadRunner® 4.0 and Astra™ SiteTest 1.0 products are provided with a script generation tool referred to as the "Vuser Generator." This tool operates by capturing actual HTTP and/or HTTPS (Secure HTTP) traffic between a standard Web browser and the Web server, and recording this traffic into a script file. Thus, for example, to provide a Vuser that repeatedly accesses a particular sequence of Web pages, the user can launch the Vuser Generator, and then sequentially access the Web pages with a standard browser.

To generate a test that emulates multiple concurrent users, the user can generate and save multiple test scripts (e.g., a set of 5 scripts), and then use a "Scenario Wizard" tool to define how these scripts will be used to test the site. For example, the user can define a group of 10 Vusers that will play back the same script, and can define other Vusers that will play back uniquely-assigned scripts. This test definition (referred to generally as a "test scenario" or "scenario") is then stored as a "scenario file" that can thereafter be loaded and run to test the site.

While the Vuser Generator tool provides a simple method for generating test scripts, the tool requires the user to actively browse the Web site. This can be burdensome, especially if the Web site is large. For example, if the Web site has 200 pages, the user would normally have to browse all 200 pages to generate a test script (or a set of test scripts) that covers the entire site. The process of generating a scenario can also be cumbersome, especially if a large number of scripts are involved.

In addition, the test scripts and scenarios generated by this method are not necessarily representative of the paths and browsing behaviors followed by typical visitors. (To distinguish between users of the disclosed tools and regular users of the site, the term "visitor" is used herein to refer to the latter.) For example, the resulting load test may heavily stress an area of the site that is rarely accessed, while failing to adequately stress more popular areas of the site.

It is thus a goal of the invention to provide a test generation tool that eliminates the need for the user to browse the Web site or actively define the scenario. It is a further goal of the invention to provide a test generation method that allows the site to be stressed according to typical usage patterns.

In accordance with these goals, a software module and associate method are provided for automatically generating test scenarios (including test scripts) based on information stored within a server access log file. The server access log file ("log file") is preferably a standard-format log file of the type commonly generated by commercially-available Web servers. These log files contain information about accesses that have been made to the site over some period of time (typically a few weeks), including the IP addresses of the visitors, the URLs (Uniform Resource Locators) that were accessed, and the times and dates of the accesses.

In accordance with the invention, the software module implements a scenario generation process that preserves the load distribution (e.g., the distribution of accesses among URLs) represented within the log file. Thus, for example, if, during the time period to which the log file corresponds, a first area of the site was accessed more heavily than a second area, the resulting test scenario will stress the first area more heavily than the second area (according to the same general proportions as within the log file).

In a preferred embodiment, the scenario generation process involves using the IP addresses and timestamps within the log file to "trace" the navigation paths taken by individual users. This produces a routes list which includes information about the number of "hits" that occurred on each link. The routes list is then translated into a scenario that comprises a set of test scripts (stored as script files) and a scenario file. The scenario can thereafter be loaded and executed, using either the LoadRunner product or the Astra SiteTest product, to load-test the Web site.

BRIEF DESCRIPTION OF THE DRAWINGS

The various features of the invention will now be described in greater detail with reference to the drawings of a preferred software package known as Astra™ SiteManager ("Astra"), its screen displays, and various related components. In these drawings, reference numbers are re-used, where appropriate, to indicate a correspondence between referenced items.

FIG. **1** is a screen display which illustrates an example Web site map generated by Astra, and which illustrates the menu, tool and filter bars of the Astra graphical user interface.

FIGS. **2** and **3** are screen displays which illustrate respective zoomed-in views of the site map of FIG. **1**.

FIG. **4** is a screen display which illustrates a split-screen display mode, wherein a graphical representation of a Web site is displayed in an upper window and a textual representation of the Web site is displayed in a lower window.

FIG. **5** is a screen display which illustrates a navigational aid of the Astra graphical user interface.

FIG. **6** is a screen display illustrating a feature which allows a user to selectively view the outbound links of URL in a hierarchical display format.

FIG. **7** is a block diagram which illustrates the general architecture of Astra, which is shown in the context of a client computer communicating with a Web site.

FIG. **8** illustrates the object model used by Astra.

FIG. **9** illustrates a multi-threaded process used by Astra for scanning and mapping Web sites.

FIG. **10** illustrates the general decision process used by Astra to scan a URL.

FIG. **11** is a block diagram which illustrates a method used by Astra to scan dynamically-generated Web pages.

FIG. **12** is a flow diagram which further illustrates the method for scanning dynamically-generated Web pages.

FIGS. **13–15** are a sequence of screen displays which further illustrate the operation of Astra's dynamic page scanning feature.

FIG. **16** is a screen display which illustrates the site map of FIG. **1** following the application of a filter which filters out all URLs (and associated links) having a status other than "OK."

FIG. **17** illustrates the general program sequence followed by Astra to generate filtered maps of the type shown in FIG. **16**.

FIG. **18** illustrates the filtered map of FIG. **16** redisplayed in Astra's Visual Web Display™ format.

FIG. **19** is a screen display which illustrates an activity monitoring feature of Astra.

FIG. **20** illustrates a decision process used by Astra to generate link activity data (of the type illustrated in FIG. **19**) from a server access log file.

FIG. **21** is a screen display which illustrates a map comparison tool of Astra.

FIG. **22** is a screen display which illustrates a link repair feature of Astra.

FIGS. **23** and **24** are partial screen displays which illustrate layout features in accordance with another embodiment of the invention.

FIG. **25** illustrates the process used by the Astra LoadRunner and SiteTest products to load-test Web sites.

FIG. **26** is a partial screen display which illustrates the user interface of the Astra SiteTest Controller.

FIGS. **27–29** are partial screen displays which illustrate an automated load-testing feature of the invention.

FIG. **30** is a flow diagram which illustrates the flow of data during the automated generation of load test scenarios.

FIGS. **31** and **32** are flow charts of a process for automatically generating load testing scenarios using information stored within a server access log file.

The screen displays included in the figures were generated from screen captures taken during the execution of the Astra SiteManager and Astra SiteTest programs. The original screen captures have been modified to comply with patent office standards.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The description of the preferred embodiment is arranged within the following sections:

I. Glossary of Terms and Acronyms
II. Overview of Astra SiteManager
III. Map Layout and Display Methodology
IV. Astra Graphical User Interface
V. Astra Software Architecture
VI. Scanning Process
VII. Scanning and Mapping of Dynamically-Generated Pages
VIII. Display of Filtered Maps
IX. Tracking and Display of Visitor Activity
X. Map Comparison Tool

I. Glossary of Terms and Acronyms

The following definitions and explanations provide background information pertaining to the technical field of the present invention, and are intended to facilitate an understanding of both the invention and the preferred embodiments thereof. Additional definitions are provided throughout the detailed description.

Internet. The Internet is a collection of interconnected public and private computer networks that are linked together by a set of standard protocols (such as TCP/IP, HTTP, FTP and Gopher) to form a global, distributed network.

Document. Generally, a collection of data that can be viewed using an application program, and that appears or is treated as a self-contained entity. Documents typically include control codes that specify how the document content is displayed by the application program. An "HTML document" is a special type of document which includes HTML (HyperText Markup Language) codes to permit the document to be viewed using a Web browser program. An HTML document that is accessible on a World Wide Web site is commonly referred to as a "Web document" or "Web page." Web documents commonly include embedded components, such as GIF (Graphics Interchange Format) files, which are represented within the HTML coding as links to other URLs. (See "HTML" and "URL" below.)

Hyperlink. A navigational link from one document to another, or from one portion (or component) of a document to another. Typically, a hyperlink is displayed as a highlighted word or phrase that can be clicked on using the mouse to jump to the associated document or document portion.

Hypertext System. A computer-based informational system in which documents (and possibly other types of data entities) are linked together via hyperlinks to form a user-navigable "web." Although the term "text". appears within "hypertext," the documents and hyperlinks of a hypertext system may (and typically do) include other forms of media. For example, a hyperlink to a sound file may be represented within a document by graphic image of an audio speaker.

World Wide Web. A distributed, global hypertext system, based on an set of standard protocols and conventions (such as HTTP and HTML, discussed below), which uses the Internet as a transport mechanism. A software program which allows users to request and view World Wide Web ("Web") documents is commonly referred to as a "Web browser," and a program which responds to such requests by returning ("serving") Web documents is commonly referred to as a "Web server."

Web Site. As used herein, "web site" refers generally to a database or other collection of inter-linked hypertextual documents ("web .documents") and associated data entities, which is accessible via a computer network, and which forms part of a larger, distributed informational system. Depending upon its context, the term may also refer to the associated hardware and/or software server components used to provide access to such documents. When used herein with initial capitalization (i.e., "Web site"), the term refers more specifically to a web site of the World Wide Web. (In general, a Web site corresponds to a particular Internet domain name, such as "merc-int.com," and includes the content of or associated with a particular organization.) Other types of web sites may include, for example, a hypertextual database of a corporate "intranet" (i.e., an internal network which uses standard Internet protocols), or a site of a hypertext system that uses document retrieval protocols other than those of the World Wide Web.

Content Object. As used herein, a data entity (document, document component, etc.) that can be selectively retrieved from a web site. In the context of the World Wide Web, common types of content objects. include HTML documents, GIF files, sound files, video files, Java applets and aglets, and downloadable applications, and each object has a unique identifier (referred to as the "URL") which specifies the location of the object. (See "URL" below.)

URL (Uniform Resource Locator). A unique address which fully specifies the location of a content object on the Internet. The general format of a URL is protocol:// machine-address/path/filename. (As will be apparent from the context in which it is used, the term "URL" is also used herein to refer to the corresponding content object itself.)

Graph/Tree. In the context of database systems, the term "graph" (or "graph structure") refers generally to a data structure that can be represented as a collection of interconnected nodes. As described below, a Web site can conveniently be represented as a graph in which each node of the graph corresponds to a content object of the Web site, and in which each interconnection between two nodes represents a link within the Web site. A "tree" is a specific type of graph structure in which exactly one path exists from a main or "root" node to each additional node of the structure. The terms "parent" and "child" are commonly used to refer to the interrelationships of nodes within a tree structure (or other hierarchical graph structure), and the term "leaf" or "leaf node" is used to refer to nodes that have no children. For additional information on graph and tree data structures, see Alfred V. Aho et al, *Data Structures and Algorithms,* Addison-Wesley, 1982.

TCP/IP (Transfer Control Protocol/Internet Protocol). A standard Internet protocol which specifies how computers exchange data over the Internet. TCP/IP is the lowest level data transfer protocol of the standard Internet protocols.

HTML (HyperText Markup Language). A standard coding convention and set of codes for attaching presentation and linking attributes to informational content within documents. During a document authoring stage, the HTML codes (referred to as "tags") are embedded within the informational content of the document. When the Web document (or "HTML document") is subsequently transmitted by a Web server to a Web browser, the codes are interpreted by the browser and used to parse and display the document. In addition to specifying how the Web browser is to display the document, HTML tags can be used create hyperlinks to other Web documents. For more information on HTML, see Ian S. Graham, *The HTML Source Book,* John Wiley and Sons, Inc., 1995 (ISBN 0471-11894-4).

HTTP (Hypertext Transfer Protocol). The standard World Wide Web client-server protocol used for the exchange of information (such as HTML documents, and client requests for such documents) between a Web browser and a Web server. HTTP includes several different types of messages which can be sent from the client to the server to request different types of server actions. For example, a "GET" message, which has the format GET <URL>, causes the server to return the content object located at the specified URL.

Webcrawling. Generally, the process of accessing and processing web site content (typically using an automated searching/parsing program) and generating a condensed representation of such content. Webcrawling routines are commonly used by commercial Internet search engines (such as Infoseek™ and Alta Vista™) to generate large indexes of the terms that appear within the various Web pages of the World Wide Web.

API (Application Program Interface). A software interface that allows application programs (or other types of programs) to share data or otherwise communicate with one another. A typical API comprises a library of API functions or "methods" which can be called in order to initiate specific types of operations.

CGI (Common Gateway Interface). A standard interface which specifies how a Web server (or possibly another information server) launches and interacts with external programs (such as a database search engine) in response to requests from clients. With CGI, the Web server can serve information which is stored in a format that is not readable by the client, and present such information in the form of a client-readable Web page. A CGI program (called a "CGI script") may be invoked, for example, when a Web user fills out an on-screen form which specifies a database query. For more information on CGI, see Ian S. Graham, *The HTML Source Book,* John Wiley and Sons, Inc., 1995 (ISBN 0471-11894-4), pp. 231–278.

OLE (Object Linking and Embedding). An object technology, implemented by Windows-based applications, which allows objects to be linked to one another and embedded within one another. OLE Automation, which is a feature of OLE 2, enables a program's functionality to be exposed as OLE objects that can be used to build other applications. For additional information on OLE and OLE Automation, see *OLE 2 Programmer's Reference Manual, Volume One,* Microsoft Corporation, 1996 (ISBN 1-55615-628-6).

II. Overview of Astra SiteManager

The Astra SiteManager program ("Astra") includes various features for facilitating the mapping, analysis (including load-testing) and management of Web sites. The program runs on a client computer under either the Windows® NT or the Windows® 95 operating system.

Given the address of a Web site's home page, Astra automatically scans the Web site and creates a graphical site map showing all of the URLs of the site and the links between these URLs. The layout and display method used by Astra for generating the site map provides a highly intuitive, graphical representation which allows the user to visualize the layout of the site. Using this mapping feature, in combination with Astra's powerful set of integrated tools for navigating, filtering and manipulating the Web site map, users can intuitively perform such actions as isolate and repair broken links, focus in on Web pages (and other content objects) of a particular content type and/or status, and highlight modifications made to a Web site since a prior mapping. In addition, users can utilize a Dynamic Scan™ feature of Astra to automatically append dynamically-generated Web pages (such as pages generated using CGI scripts) to their maps. In addition, using Astra's activity monitoring features, users can monitor visitor activity levels on individual links and URLs, and study visitor behavior patterns during Web site visits. Further, the user can invoke a Load Wizard™ feature to cause the automatic generation of a load-testing scenario, which can in-turn be used to load-test the site using Mercury Interactive's LoadRunner and Astra SiteTest products.

Astra is based on a highly extensible architecture which facilitates the addition of new tools to the Astra framework. As part of this architecture, a "core" Astra component (which includes the basic Web site scanning and mapping functionality) has an API for supporting the addition of plug-in components. This API includes functions for allowing the plug-in components to manipulate the display of the site map, and to display their own respective data in conjunction with the Astra site map. Through this API, new applications can be added which extend the functionality of the package while taking advantage of the Astra mapping scheme.

Throughout this description, names of product features and software components are used with initial capitalization. These names are used herein for ease of description only, and are not intended to limit the scope of the invention.

FIGS. **1–3** illustrate Astra's primary layout methodology, referred to herein as "Visual Web Display™," for displaying graphical representations ("site maps" or "maps") of Web sites. These figures will also be used to describe some of the graphical user interface (GUI) features of Astra.

FIG. **1** illustrates a site map **30** of a demonstration Web site which was derived from the actual Web site of Mercury Interactive (i.e., the URLs accessible under the "merc-int.com" Internet domain name). (For purposes of this detailed description, it may be assumed that "Web site" refers to the content associated with a particular Internet domain name.) The Web site is depicted by Astra as a collection of nodes, with pairs of nodes interconnected by lines. Each node of the map represents a respective content object of the Web site and corresponds to a respective URL. (The term "URL" is used herein to refer interchangeably to both the address of the content object and to the object itself; where a distinction between the two is important, the term "URL" is followed by an explanatory parenthetical.) Examples of URLs (content objects) which may exist within a typical Web site include HTML documents (also referred to herein as "Web pages"), image files (e.g., GIF and PCX files), mail messages, Java applets and aglets, audio files, video files, and applications.

As generally illustrated by FIGS. **3** and **4**, different icons are used to represent the different URL types when the nodes are viewed in a sufficiently zoomed-in mode. (Generic icons of the type best illustrated by FIG. **18** are used to display nodes that fall below a predetermined size threshold.) As described below, special icons and visual representations are also used to indicate status information with respect to the URLs. For example, special icons are used to depict, respectively, inaccessible URLs, URLs which are missing, URLs for which access was denied by the server, and URLs which have been detected but have not been scanned. (The term "scan" refers generally to the process of sending informational requests to server components of a computer network, and in the context of the preferred embodiment, refers to the process of sending requests to Web server

components to obtain Web site content associated with specific URLs.)

The lines which interconnect the nodes (URL icons) in FIGS. 1–3 (and the subsequent figures with screen displays) represent links between URLs. As is well understood in the art, the functions performed by these links vary according to URL type. For example, a link from one HTML document to another HTML document normally represents a hyperlink which allows the user to jump from one document to the other while navigating the Web site with a browser. In FIG. 1, an example of a hyperlink which links the home page URL (shown at the center of the map) to another HTML page (displayed to the right of the home page) is denoted by reference number 32. (As generally illustrated in FIG. 1 and the other figures which illustrate screen displays, regular HTML documents are displayed by Astra as a shaded document having text thereon.) A link between an HTML document and a GIF file, such as link 36 in FIG. 3, normally represents a graphic which is embedded within the Web page.

Maps of the type illustrated in FIG. 1 are generated by Astra using an HTTP-level scanning process (described below) which involves the reading and parsing the Web site's HTML pages to identify the architecture (i.e., the arrangement of URLs and links) of the Web site, and to obtain various status information (described below) about the Web site's URLs. The basic scanning process used for this purpose is generally similar to the scanning process used by conventional Webcrawlers. As part of Astra's Dynamic Scan feature, Astra additionally implements a special dynamic page scanning process which permits dynamically-generated Web pages to be scanned and included in the Web site map. As described below, this process involves capturing the output of a Web browser when the user submits an HTML-embedded form (such as when the user submits a database query), and then reusing the captured dataset during the scanning process to recreate the form submission and append the results to the map.

Table 1 lists the predefined icons that are used by Astra to graphically represent different URL types within site maps. As illustrated, the URL icons generally fall into two categories: object-type ("URL type") icons and status icons. The object-type icons are used to indicate the content or service type of URLs that have been successfully scanned. The status icons are used to indicate the scanning status (not found, access denied, etc.) of URLs for which either (i) scanning has not been performed, or (ii) scanning was unsuccessful. Various examples of these two types of icons are included in the figures.

TABLE 1

URL Type	Scanning Status
HTML	Not found
HTML with Form	Not Scanned
Image	Inaccessible
Sound	Access Denied
Application	
Text	
Unknown	
Video	
Gopher	
FTP	
Dynamic Page	

Once the map has been generated, the user can interactively navigate the map using various navigation tools of the Astra GUI, such as the zoom-in and zoom-out buttons 34, 36 (FIG. 1) and the scrolling controls 40, 42 (FIGS. 2 and 3). To zoom-in on a particular region of the map 30, the user can click on the zoom-in button 34 and then use the mouse to draw a box around the map region of interest; Astra will then re-size the highlighted region to generally fit the display screen. As will be recognized by those skilled in the art, the ability to zoom in and out between high level, perspective views which reveal the overall architecture of the site, and magnified (zoomed-in) sub-views which reveal URL-specific information about the Web site, greatly facilitates the task of navigating and monitoring Web site content.

As generally illustrated by FIG. 3, the annotations (page titles, filenames, etc.) of the URLs begin to appear (below the associated icons) as the user continues to zoom in. As further illustrated by FIG. 3, the URL (address) of a node is displayed when the mouse cursor is positioned over the corresponding icon.

While navigating the map, the user can retrieve a URL (content object) from the server by double-clicking on the corresponding URL icon; this causes Astra to launch the client computer's default Web browser (if not already running), which in-turn retrieves the URL from the Web server. For example, the user can double-click on the URL icon for an HTML document (using the left mouse button) to retrieve and view the corresponding Web page. When the user clicks on a URL icon using the right mouse button, a menu appears which allows the user to perform a variety of actions with respect to the URL, including viewing the URL's properties, and launching an HTML editor to retrieve and edit the URL. With reference to FIG. 3, for example, the user can click on node 44 (using the right mouse button), and can then launch an HTML editor to edit the HTML document and delete the reference to missing URL 45. (As illustrated by FIG. 3, missing URLs are represented within Astra maps by a question mark icon.)

One important feature of Astra, referred to herein as "Automatic Update," allows the user to update an existing Web site map to reflect any changes that have been made to the map since a prior mapping of the site. To initiate this feature, the user selects a "start Automatic Update" button 37 (FIG. 1), or selects the corresponding menu item, while viewing a site map. This initiates a re-scanning process in which Astra scans the URLs of the Web site and updates the map data structure to reflect the current architecture of the site. As part of this process, Astra implements a caching protocol which eliminates the need to download URLs and URL headers that have not been modified since the most recent mapping. (This protocol is described below under the heading "SCANNING PROCESS.") This typically allows the map to be updated in a much shorter period of time than is required to perform the original mapping. This feature is particularly useful for Webmasters of large Web sites that have dynamically-changing content.

Other features of Astra are described in the following sections.

III. May Layout and Display Methodology (FIGS. 1–3, 23 and 24)

An important aspect of the invention is the methodology used by Astra for presenting the user with a graphical, navigable representation of the Web site. This feature of Astra, which is referred to as Visual Web Display (abbreviated as "VWD" herein), allows the user to, view and navigate complex Web structures while visualizing the inter-relationships between the data entities of such structures. The method used by Astra to generate VWD site maps is referred to herein as the "Solar Layout method," and is described at the end of this section.

One aspect of the VWD format is the manner in which children nodes ("children") are displayed relative to their

respective parent nodes ("parents"). (In the context of the preferred embodiment, the term "node" refers generally to a URL icon as displayed within the site map.) As illustrated by the collection of nodes shown in FIG. **3**, the parent **44** is displayed in the center of the cluster, and the seven children **48** are positioned around the parent **44** over an angular range of 360 degrees. One benefit of this layout pattern is that it allows collections of related nodes to be grouped together on the screen in relatively close proximity to one another, making it easy for the user identify the parent-child relationships of the nodes. This is in contrast to the expandable folder type representations used by Webmapper™, the Windows® 95 Explorer, and other Windows® applications, in which it is common for a child to be separated from its parent folder by a long list of other children.

In this FIG. **3** example, all of the children **48** are leaf nodes (i.e., nodes which do not themselves have children). As a result, all of the children **48** are positioned approximately equidistant from the parent **44**, and are spaced apart from one another by substantially equal angular increments. Similar graphical representations to that of FIG. **3** are illustrated in FIG. **1** by node clusters **52**, **54** and **56**. As illustrated by these three clusters in FIG. **1**, both (i) the size of parent icon, and (ii) the distance from the parent to its children, are proportional to the number of immediate children of the parent. Thus, for example, cluster **56** has a larger diameter (and a larger parent icon) than clusters **52** and **54**. This has the desirable effect of emphasizing the pages of the Web site that have the largest numbers of outgoing links. (As used herein, the term "outgoing links" includes links to GIF files and other embedded components of document.)

As best illustrated by cluster **64** in FIG. **2**, of which node **65** is the primary parent or "root" node, children which have two or more of their own children (i.e., grandchildren of the root) are positioned at a greater distance from the root node **65** than the leaf nodes of the cluster, with this distance being generally proportional to the size of the sub-cluster of which the child is the parent. For example, node **66** (which has **3** children) is positioned farther from the cluster's root node **65** than leaf nodes **70**; and the parent of cluster **60** is positioned farther from the root node **65** than node **66**. As illustrated in FIG. **1**, this layout principal is advantageously applied to all of the nodes of the Web site that have children. The recursive method (referred to as "Solar Layout") used by Astra to implement these layout and display principles is described below.

Another aspect of the layout method is that the largest "satellite" cluster of a parent node is centered generally opposite from (along the same line as) the incoming link to the parent node. This is illustrated, for example, by cluster **54** in FIG. **1** and by cluster **60** in FIG. **2**, both of which are positioned along the same line as their respective parents. This aspect of the layout arrangement tends to facilitate visualization by the user of the overall architecture of the site.

As will be apparent from an observation of FIG. **1**, the graphical map produced by the application of the above layout and display principles has a layout which resembles the general arrangement of a solar system, with the home page positioned as the sun, the children of the home page being in orbit around the sun, the grandchildren of the home page being in orbit around their immediate respective parents, and so on. One benefit of this mapping arrangement is that it is well suited for displaying the entire site map of a complex Web site on a single display screen (as illustrated in FIG. **1**). Another benefit is that it provides an intuitive structure for navigating the URLs of a complex Web site. While this mapping methodology is particularly useful for the mapping of Web sites, the methodology can also be applied, with the realization of similar benefits, to the mapping of other types of databases. For example, the VWD methodology can be used to facilitate the viewing and navigation of a conventional PC file system.

Another benefit of this site map layout and display methodology is that the resulting display structure is well suited for the overlaying of information on the map. Astra takes full advantage of this benefit by providing a set of API functions which allow other applications (Astra plug-ins) to manipulate and add their respective display data to the site map. An example of an Astra plug-in which utilizes this feature is the Action Tracker™ tool, which superimposes user activity data onto the site map based on analyses of server access log files. The Astra plug-in API and the Action Tracker plug-in are described in detail below.

As illustrated in FIG. **1**, all of the nodes of the site map (with the exception of the home page node) are displayed as having a single incoming link, even though some of the URLs of the depicted Web site actually have multiple incoming links. Stated differently, the Web site is depicted in the site map **30** as though the URLs are arranged within a tree data structure (with the home page as the main root), even though a tree data structure is not actually used. This simplification to the Web site architecture is made by extracting a span tree from the actual Web site architecture prior to the application of a recursive layout algorithm, and then displaying only those links which are part of the spanning tree. (In applications in which the database being mapped is already arranged within a tree directory structure, this step can be omitted.) As a result, each URL of the Web site is displayed exactly once in the site map. Thus, for example, even though a particular GIF file may be embedded within many different pages of the Web site, the GIF file will appear only once within the map. This simplification to the Web site architecture for mapping purposes makes it practical and feasible to graphically map, navigate and analyze complex Web sites in the manner described above.

Because the Visual Web Display format does not show all of the links of the Web site, Astra supports two additional display formats which enable the user to display, respectively, all of the incoming links and all of the outgoing links of a selected node. To display all of the outgoing links of a given node, the user selects the node with the mouse and then selects the "display outgoing links" button **72** (FIG. **1**) from the tool bar **46**. Astra then displays a hierarchical view (in the general form of a tree) of the selected node and its outgoing links, as illustrated by FIG. **6**. Similarly, to display the incoming links of a node, the user selects the node and then clicks on the "display incoming links" button **71**. (A screen display illustrating the incoming links format is shown in FIG. **22**.) To restore the Visual Web Display view, the user clicks on the VWD button **73**.

The Solar Layout method (used to generate VWD-format site maps) generally consists of three steps, the second two of which are performed recursively on a node-by-node basis. These three steps are outlined below, together with associated pseudocode representations. In addition, a source code listing of the method (in C++) is included in microfiche appendix as Appendix A.

Step 1—Select Span Tree

In this step, a span tree is extracted from the graph data structure which represents the arrangement of nodes and links of the Web site. (The graph data structure is implemented as a "Site Graph" OLE object, as described below.)

Any standard span tree algorithm can be used for this purpose. In the preferred embodiment, a shortest-path span tree algorithm known as "Dijkstra's algorithm" is used, as implemented within the commercially-available LEDA (Library of Efficient Data types and Algorithms) software package. As applied within Astra, this algorithm finds the shortest paths from a main root node (corresponding to the Web site's home page or some other user-specified starting point) to all other nodes of the graph structure. The result of this step is a tree data structure which includes all of the URLs of the graph data structure with the home page represented as the main root of the tree. For examples of other span tree algorithms which can be used, see Alfred V. Aho et al, *Data Structures and Algorithms,* Addison-Wesley, 1982.)

Step 2—Solar Plan

This is a recursive step which is applied on a node-by-node basis in order to determine (i) the display size of each node, (ii) the angular spacings for positioning the children nodes around their respective parents, and (iii) the distances for spacing the children from their respective parents. For each parent node, the respective sizes of the parent's satellites are initially determined. (A "satellite" is any child of the parent plus the child's descendants, if any.) The satellite sizes are then used to allocate (a) angular spacings for positioning the satellites around the parent, and (b) the radial distances between the satellites and the parent. This process is repeated for each parent node (starting with the lower level parent nodes and working up toward the home page) until all nodes of the graph have been processed. The following is a pseudocode representation of this process:

```
Node::SolarPlan()
{
IF node has no children
        return basic graphical dimension for a single node
ELSE
        For each linked node as selected in the span tree, call SolarPlan()
        recursively;
        Based on the sum of the sizes of the satellites, allocate angle for
        positioning satellites around parent, and set satellite distances from
        parent;
        Calculate size of present cluster (parent plus satellites).
}
```

A modified Solar Plan process which incorporates two additional layout features is described below and illustrated by FIGS. **23** and **24**,

Step 3—Solar Place

This step recursively positions the nodes on the display screen, and is implemented after Step 2 has been applied to all of the nodes of the graph. The sequence starts by positioning the home page at the center, and then uses the angle and distance settings calculated in Step 2 to position the children of the home page around the home page. This process is repeated recursively for each parent node until all of the nodes have been positioned on the screen.

```
Node::SolarPlace(x, y, entry_angle)
{
Move this node to location (x,y)
For each satellite:
        calculate final angle as the sum of the entry_angle and the angle
        allocated in Step 2; Calculate satellite center (x and y coordinates)
        based on new angle and distance from current node; Call SolarPlace
        using the above-calculated angle and location.
}
```

In the above pseudocode representation, the "x" and "y" parameters specify the screen position for the placement of a node (icon), and the "entry_angle" parameter specifies the angle of the line (link) between the node and its respective parent. In the preferred embodiment, the method is implemented such that the largest satellite of a parent node is positioned using the same entry angle as the parent node, so that the satellite center, parent node, and parent of the parent node all fall generally along the same line. (The determination of the largest satellite is performed in Step 2.) As indicated above, this aspect of the layout method is illustrated in FIG. **1** by cluster (satellite) **54**, which is positioned along the same line as both its immediate parent icon and the home page icon.

A modified Solar Plan process will now be described with reference to the screen displays of FIGS. **23** and **24**, and to the corresponding pseudocode representation below. This modified process incorporates two additional layout features which relate to the positioning of the satellites around a parent. These layout features are implemented within the attached source code listing (Appendix A), and are represented generally by the highlighted text of the following pseudocode sequence:

```
Node::SolarPlan()
{
IF node has no children
        return basic graphical dimension for a single node
ELSE
        For each linked node as selected in the span tree, call SolarPlan()
        recursively;
        Based on the sum of the sizes of the satellites + minimal weight of
        the incoming link, allocate angle for positioning satellites around
        parent, and set satellite distances from parent;
        Sort satellite list as follows: smallest child first, and in jumps of
        two next child up to the biggest, and then back to second biggest
        and in jumps of two down to smallest (e.g., 1, 3, 5 ... biggest,
        second biggest, ... 6, 4, 2);
        Calculate size of present cluster (parent plus satellites).
}
```

The first of the two layout features is illustrated by FIG. **23**, which is a partial screen display (together with associated annotations) of a parent-child cluster comprising a parent **79** and seven children or satellites **75**. This layout feature involves allocating an angular interval (e.g., 20 degrees) to the incoming link **81** to the parent **79**, and then angularly spacing the satellites **75** (which in this example are all leaf nodes) over the remaining angular range. In the preferred embodiment, this is accomplished by assigning a minimal weight (corresponding to the angular interval) to the incoming link **81**, and then treating this link **81** as one of the outgoing links **83** when assigning angular positions to the satellites **75**. As a result of this step, the satellites **75** are positioned around the parent **79** over an angular range of less than 360 degrees—in contrast to the clusters of FIGS. **1–5**, in which the satellites are positioned over the full 360 range. (In this FIG. **23** example, because all of the satellites **75** are leaf nodes, the satellites **75** are positioned equidistant from the parent **79** with equal angular spacings.) One benefit of

this added step is that it allows the user to more easily distinguish the incoming link **81** to a parent **79** from the outgoing links **83** from the parent. With reference to the angular notations of FIG. **23**, the minimal weight is preferably selected such that the angle θ_1 between the incoming link **81** and each of the two adjacent parent-child links **83** is greater than or equal to the minimum angle θ_2 between adjacent parent-child links **83** for the given cluster. This layout feature is also illustrated by FIG. **24**.

The second of the two additional layout features involves ordering the satellites around the parent based on the respective sizes of the satellites. This feature comes into play when a parent node has multiple satellites that differ in size from one another. The layout arrangement which is produced by this feature is generally illustrated by FIG. **24**, which shows a cluster having a parent node (labeled "CNN SHOWBIZ") and 49 satellites. As illustrated by this screen image, the satellites are ordered such that the smallest satellites **85** are angularly positioned closest to the incoming link **89** to the parent, and such that the largest satellites **91**A–E are positioned generally opposite from the incoming link **89**. This is preferably accomplished by sorting the satellites using the sorting algorithm of the above pseudocode sequence (which produces a sorted satellite list in which the satellites progress upward from smallest to largest, and then progress downward from second largest to second smallest), and then positioning the satellites around the parent (starting at the incoming link **89**) in the order which results from the sorting process. In this example, the largest satellite **91**A is positioned opposite the incoming link **89**; the second and third largest satellites **91**B and **91**C are positioned adjacent to the largest satellite **91**A; the fourth and fifth largest satellites **91**D and **91**E are positioned adjacent to the second and third largest satellites **91**B and **91**C (respectively); and so on. As is apparent from FIG. **24**, this layout feature tends to produce a highly symmetrical layout.

Other aspects of the Solar Layout method will be apparent from an observation of the screen displays and from the source code listing of Appendix A.

V. Astra Graphical User Interface (FIGS. 1 and 4–6)

As illustrated in FIG. **1**, the Astra menu bar includes seven menu headings: FILE, VIEW, SCAN, MAP, URL, TOOLS and HELP. From the FILE menu the user can perform various file-related operations, such as save a map file to disk or open a previously generated map file. From the VIEW menu the user can select various display options of the Astra GUI. From the SCAN menu the user can control various scanning-related activities, such as initiate or pause the automatic updating of a map, or initiate a dynamic page scan session. From the MAP menu, the user can manipulate the display of the map, by, for example, collapsing (hiding) all leaf nodes, or selecting the Visual Web Display mode. From the URL menu, the user can perform operations with respect to user-selected URLs, such as display the URL's content with a browser, invoke an editor to modify the URL's content, and display the incoming or outgoing links to/from the URL.

From the TOOLS menu the user can invoke various analysis and management related tools. For example, the user can invoke a map comparison tool which generates a graphical comparison between two maps. This tool is particularly useful for allowing the user to readily identify any changes that have been made to a Web site's content since a previous mapping. The user can also invoke the Action Tracker tool, which superimposes link activity data on the Web site map to allow the user to readily ascertain the links and URLs that have the most hits. The Action Tracker tool also includes code for generating load testing scenarios from the link activity data. The user can also invoke a Link Doctor tool which facilitates the repairing of broken links. These and other tools and features of Astra are described in the subsequent sections.

With further reference to FIG. **1**, the Astra GUI includes a tool bar **46** and a filter bar **47**, both of which can be selectively displayed as rieeded. The tool bar **46** includes buttons for initiating commonly-performed operations. From left to right in FIG. **1**, these functions are as follows: (a) start generation of new map, (b) open map file, (c) save map to disk, (d) print, (e) size map to fit within window, (f) zoom in, (g) zoom out, (h) display incoming links of selected node; (i) display outgoing links of selected node, (j) display map in Visual Web Display format, (k) initiate Automatic Update, (l) pause Automatic. Update, (m) resume Automatic Update, (n) initiate Dynamic Scan, and (o) stop Dynamic Scan. (The function performed by each button is indicated textually when the mouse cursor is positioned over the respective button.)

The filter bar **47** includes a variety of different filter buttons for filtering the content of site maps. When the user clicks on a filter button, Astra automatically hides all links and pages of a particular type or status, as illustrated in FIG. **16** and discussed below. The filter buttons are generally divided into three groups: content/service filters **49**, status filters **50**, and location filters **51**. From left to right in FIG. **1**, the content/service filters **49** filter out URLs of the following content or service types: (a) HTML, (b) HTML forms, (c) images, (d) audio, (e) CGI, (f) Java, (g) other applications, (h) plain text, (i) unknown, (j) redirect, (k) video, (l) Gopher, (m) FTP, and (n) all other Internet services. The status filters **50** filter out URLs of the following statuses (from left to right): (a) not found, (b) inaccessible (e.g., no response from server), (c) access denied, (d) not scanned, and (e) OK. The left-hand and right-hand location filters **51** filter out local URLs and external URLs, respectively, based on the domain names of the URLs. Multiple filters can be applied concurrently.

FIG. **4** illustrates a split-screen mode which allows the user to view a graphical representation of the Web site in an upper window **76** while viewing a corresponding textual representation (referred to as "List View") in a lower window **78**. To expose the List View window **78**, the user drags and drops the separation bar **80** to the desired position on the screen. Each line of text displayed in the List View window **78** represents one node of the site map, and includes various information about the node. For each node, this information includes: the URL (i.e., address), an annotation, the scanning status (OK, not found, inaccessible, etc.), the associated communications protocol (HTTP, mailto, FTP, etc.), the content type, the file size (known only if the entire file has been retrieved), the numbers of inbound links and outbound links, and the date and time of last modification. (The outbound link and last modification information can be exposed in the FIG. **4** screen display by dragging the horizontal scrolling control **77** to the right.)

As described below, this information about the nodes is obtained by Astra during the scanning process, and is stored in the same data structure **114** (FIG. **9**) that is used to build the map. As additionally described below, whenever the user initiates an Automatic Update, Astra uses the date/time of last modification information stored locally in association with each previously-mapped HTML document to determine whether the document needs to be retrieved and parsed. (The parsing process is used to identify links to other URLs, and to identify other HTML elements relevant to the

mapping process.) As indicated above, this provides the significant advantage of allowing the Web site to be re-mapped without having to repeat the entire scanning/parsing process.

With further reference to FIG. **4**, whenever the user selects a node in the upper window **76**, the corresponding line in the List View window **78** is automatically highlighted. (As illustrated by node **84** in FIG. **4**, Astra graphically represents the selection of a node by outlining the node's icon in black.) Likewise, whenever the user selects a line in the List View window **78**, the corresponding node is automatically highlighted in the upper window **76**. This feature allows the user to rapidly and efficiently associate each textual line with its graphical counterpart, and vice versa. In addition, by clicking on the headers **82** of the separation bar **80**, the user can view the listed URLs in a sorted order. For example, if the user clicks on the "in links" header, Astra will automatically sort the list of URLs according to the number of incoming links, and then display the sorted listing in the List View window **78**.

FIG. **5** illustrates a Pan Window feature of Astra. This feature facilitates navigation of the site map while in a zoomed-in mode by presenting the user with a perspective view of the navigational position within the map. To display the Pan Window **86**, the user selects the "Pan Window" menu option from the VIEW menu while viewing a map. Within the Pan Window, the user is presented with a display of the entire map **30**, with a dashed box **87** indicating the portion of the map that corresponds to the zoomed-in screen display. As the user navigates the site map (using the scrolling controls **40**, **42** and/or other navigational controls), the dashed box automatically moves along the map to track the zoomed-in screen display. The user can also scroll through the map by simply dragging the dashed box **87** with the mouse. In the preferred embodiment, the Pan Window feature is implemented in-part using a commercially-available from Stingray™ Corporation called SEC++, which is designed to facilitate the zoomed-in viewing of a general purpose graphic image.

FIG. **6** illustrates the general display format used by Astra for displaying the outgoing links of a selected node **88**. To display a node's outgoing links, the user selects the node with the mouse and then clicks on the "show outgoing links" button **72** on the tool bar. As illustrated, Astra then displays all outgoing links from the node (including any links that do not appear in the VWD site map), and displays additional levels of outgoing links (if any) which emanate from the children of the selected node. The display format used for this purpose is in the general format of a tree, with the selected node displayed as the root of the tree. An analogous display format (illustrated in FIG. **22**) is used for displaying the incoming links to a node.

V. Astra Software Architecture (FIGS. **7** and **8**)

FIG. **7** pictorially illustrates the general architecture of Astra, as installed on a client computer **92**. As illustrated, the architecture generally consists of a core Astra component **94** which communicates with a variety of different Astra plug-in applications **96** via a plug-in API **98**. The Astra core **94** includes the basic functionality for the scanning and mapping of Web sites, and includes the above-described GUI features for facilitating navigation of Web site maps. Through the plug-in API **98**, the Astra a core **94** provides an extensible framework for allowing new applications to be written which extend the basic functionality of the Astra core. As described below, the architecture is structured such that the plug-in applications can make extensive use of Astra site maps to display plug-in specific information.

The Astra plug-ins **96** and API **98** are based on OLE Automation technology, which provides facilities for allowing the plug-in components to publish information to other objects via the operating system registry (not shown). (The "registry" is a database used under the Windows® 95 and Windows® NT operating systems to store configuration information about a computer, including information about Windows-based applications installed on the computer.) At start-up, the Astra core **94** reads the registry to identify the Astra plug-ins that are currently installed on the client computer **92**, and then uses this information to launch the installed plug-ins.

In a preferred implementation, the architecture includes five Astra plug-ins: Link Doctor, Action Tracker, Test World, Load Wizard and Search Meter. The functions performed by these plug-ins are summarized by Table 2. Other applications which will normally be installed on the client computer in conjunction with Astra include a standard Web browser (FIGS. **11** and **12**), and one or more editors (not shown) for editing URL content.

TABLE 2

PLUG-IN	FUNCTION PERFORMED
Link Doctor	Fixes broken links automatically
Action Tracker	Retrieves and evaluates server access log files to generate Web site activity data (such as activity levels on individual links), and superimposes such data on site map in a user-adjustable manner.
Test World	Generates and drives tests automatically
Load Wizard	Utilizes Action Tracker activity data to automatically generate test scenarios for the load-testing of Web sites with Mercury Interactive's LoadRunner ™ and SiteTest ™ software packages. (In the implementation described herein, the Load Wizard functionality is included within the Action Tracker plug-in.)
Search Meter	Displays search engine results visually.

The Astra API allows external client applications, such as the plug-in applications **96** shown in FIG. **7**, to communicate with the Astra core **94** in order to form a variety of tasks. Via this API, client applications can perform the following types of operations:

1. Superimpose graphical information on the site map;
2. Access information gathered by the Astra scanning engine in order to generate Web site statistics;
3. Attach custom attributes to the site map, and to individual nodes and links of the site map;
4. Access some or all of a Web page's contents (HTML) during the Web site scanning process;
5. Embed the Astra GUI within the client application;
6. Add menu items to the Astra menu; and
7. Obtain access to network functionality.

The specific objects and methods associated with the API are discussed below with reference to FIG. **8**. In addition, a complete listing of the API is included in the microfiche appendix as Appendix B.

During the Web site scanning process, the Astra core **94** communicates over the Internet **110** (or an intranet) with the one or more Web server applications **112** ("Web servers") which make up the subject Web site **113**. The Web servers **112** may, for example, run on a single computer, run on multiple computers located at a single geographic location (which may, but need not, be the location of the client computer **92**), or run on multiple computers that are geo-

graphically distributed. In addition, the Web servers **112** of the Web site **113** may be virtually distributed across multiple Internet domains.

As is conventional with Internet applications, the Astra core **94** uses the TCP/IP layer **108** of the computer's operating system to communicate with the Web site **113**. Any one or more of the Astra plug-ins **96** may also use the TCP/IP layer **108** to communicate with the Web site **113**. In the preferred embodiment, for example, the Action Tracker plug-in communicates with the Web sites (via the Astra plug-in API) to retrieve server access log files for performing Web site activity analyses.

FIG. **8** illustrates the object model used by the Astra API. As illustrated, the model includes six classes of objects, all of which are implemented as OLE Automation objects. By name, the six object classes are Astra, Site Graph, Edges, Edge, Nodes, and Node. The Astra object **94** is an application object, and corresponds generally to the Astra core **94** shown in FIG. **7**. The Astra object **94** accesses and manipulates data stored by a Site Graph object **114**. Each Site Graph object corresponds generally to a map of a Web site, and includes information about the URLs and links (including links not displayed in the Visual Web Display view) of the Web site. The site-specific data stored by the Site Graph object **114** is contained within and managed by the Edges, Edge, Nodes and Nodes objects, which are subclasses of the Graph object.

Each Node object **115** represents a respective node (URL) of the site map, and each Edge object **116** represents a respective link between two URLs (nodes) of the map. Associated with each Node object and each Edge object is a set of attributes (not shown), including display attributes which specify how the respective object is to be represented graphically within the site map. For example, each Node object and each Edge object include respective attributes for specifying the color, visibility, size, screen position, and an annotation for the display of the object. These attributes can be manipulated via API calls to the methods supported by these objects **115, 116**. For example, the Astra plug-ins (FIG. **7**) can manipulate the visibility attributes of the Edge objects to selectively hide the corresponding links on the screen. (This feature is illustrated below in the description of the Action Tracker plug-in.) In addition, the Astra API includes methods for allowing the plug-ins to define and -attach custom attributes to the Edge and Node objects.

The Nodes and Edges objects **118, 119** are container objects which represent collections of Node objects **115** and Edge objects **116**, respectively. Any criterion can be used by the applications for grouping together Node objects and Edge objects. As depicted in FIG. **8**, a single Graph object **114** may include multiple Nodes objects **118** and multiple Edges objects **119**.

The methods of the Astra plug-in API generally fall into five functional categories. These categories, and the objects to which the associated methods apply, are listed below. Additional information on these methods is provided in the API listing in Appendix B.

ASTRA GUI METHODS. These methods control various aspects of the Astra GUI, such as adding, deleting, enabling and disabling Astra menu items. Supporting objects: Astra, Site Graph.

GROUPING AND ACCESS METHODS. These methods permit groupings of nodes and links to be formed, and permit the nodes and links within these groups to be accessed. Supporting objects: Site Graph, Nodes, Edges.

NODE/EDGE APPEARANCE METHODS. These methods provide control over display attributes (visibility, color, etc.) of links and nodes of the map. Supporting Objects: Node, Edge.

ATTRIBUTE ATTACHMENT METHODS. These methods permit the attachment of custom information to specific objects, and provide access to such information. Supporting objects: Site Graph, Node, Edge. Example use: Number of "hits" displayed by Action Tracker.

SCAN-TIME CONTENT ACCESS METHODS. These methods provide access by applications to Web page content retrieved during the scanning process. Supporting Objects: Site Graph, Node. Example use: At scan time, textual content of each page is passed to a spell checker application to perform a site-wide spell check.

As will be appreciated from the foregoing, the Astra architecture provides a highly extensible mapping framework which can be extended in functionality by the addition of new plug-ins applications. Additional aspects of the architecture are specified in the API description of Appendix B.

VI. Scanning Process (FIGS. **9** and **10**)

As will be apparent, the terms "node" and "link" are used in portions of the remaining description to refer to their corresponding object representations—the Node object and the Edge object.

The multi-threaded scanning process used by the Astra core **94** for scanning and mapping a Web site will now be described with reference to FIGS. **9** and **10**. As depicted in FIG. **9**, Astra uses two types of threads to scan and map the Web site: a main thread **122** and multiple lower-level scanning threads **122**. The use of multiple scanning threads provides the significant benefit of allowing multiple server requests to be pending simultaneously, which in-turn reduces the time required to complete the scanning process. A task manager process (not shown) handles issues related to the management of the threads, including the synchronization of the scanning threads **120** to the main thread **120**, and the allocation of scanning threads **122** to operating system threads.

The main thread **120** is responsible for launching the scanning threads **122** on a URL-by-URL basis, and uses the URL-specific information returned by the scanning threads **122** to populate the Site Graph object **114** ("Site Graph") with the nodes, links, and associated information about the Web site **113**. In addition, as pictorially illustrated by the graph and map symbols in box **114**, the main thread **120** periodically applies the Solar Layout method to the nodes and links of the Site Graph **114** to generate a map data structure which represents the Visual Web Display map of the Web site. (As described below, this map data structure is generated by manipulating the display attributes of the Node objects and Edge objects, and does not actually involve the generation of a separate data structure.)

Upon initiation of the scanning process by the user, the main thread **120** obtains the URL (address) of the home page (or the URL of some other starting location) of the Web site to be scanned. If the scanning process is initiated by selecting the "Automatic Update" option, the main thread **120** obtains this URL from the previously-generated Site Graph **114**. Otherwise, the user is prompted to manually enter the URL of the home page.

Once the home page URL has been obtained, the main thread **120** launches a scanning thread **122** to scan the HTML home page. As the HTML document is returned, the scanning thread **122** parses the HTML to identify links to other URLs, and to identify other predetermined HTML elements (such as embedded forms) used by Astra. (As

described below with reference to FIG. **10**, if an Automatic Update is being performed, the scanning thread downloads the home page only if the page has been modified since the last scanning of the URL; if no download of the page is required, this outgoing link information is extracted from the previously-generated Site Graph **114**.) In addition, the scanning thread **122** extracts certain information from the header of the HTML document, including the date/time of last modification, and the other information displayed in the List View window **78** of FIG. **4**. The link and header information extracted by the scanning thread **122** is represented in FIG. **9** by one of the boxes **130** labeled "URL data."

Upon completion, the scanning thread **122** notifies the main thread **120** that it has finished scanning the home page. The main thread then reads the URL data extracted by the scanning thread **122** and stores this data in the Site Graph **114** in association with a Node object which represents the home page URL. In addition, for each internal link (i.e., link to a URL within the same Internet domain) identified by the scanning thread **122**, the main thread **120** creates (or updates) a corresponding Edge object and a corresponding Node object within the Site Graph **114**, and launches a new scanning thread **122** to read the identified URL. (Edge and Node objects are also created for links to external URLs, but these external URLs are not scanned in the default mode.) These newly-launched scanning threads then proceed to scan their respective URLs in the same manner as described above (with the exception that no downloading and parsing is performed when the subject URL is a non-HTML file). Thus, scanning threads **122** are launched on a URL-by-URL basis until either all of the URLs of the site have been scanned or the user halts the scanning process. Following the completion of the scanning process, the Site Graph **114** fully represents the site map of the Web site, and contains the various URL-specific information displayed in the Astra List View window **78** (FIG. **4**). When the user saves a site map via the Astra GUI, the Site Graph **114** is written to disk.

In a default mode, links to external URLs detected during the scanning process are displayed in the site map using the "not scanned" icon (**192** in FIG. **13**), indicating that these URLs have not been verified. If the user selects a "verify external links" scanning option prior to initiating the scanning process, Astra will automatically scan these external URLs and update the map accordingly.

As part of the HTML parsing process, the scanning threads **122** detect any forms that are embedded within the HTML documents. (As described below, these forms are commonly used to allow the user to initiate back-end database queries which result in the dynamic generation of Web pages.) When a form is detected during an Automatic Update operation, the main thread **120** checks the Site Graph **114** to determine whether one or more datasets (captured by Astra as part of the Dynamic Scan feature) have been stored in association with the HTML document. For each dataset detected, Astra performs a dynamic page scan operation which involves the submission of the dataset to the URL specified within the form. This feature is further described below under the heading SCANNING AND MAPPING OF DYNAMICALLY-GENERATED PAGES.

Once the entire Web site has **113** been scanned, the Site Graph **114** represents the architecture of the Web site, including all of the detected URLs and links of the site. (If the user pauses the scanning process prior to completion, the Site Graph and VWD map represent a scanned subset of the Web site.) As described above, this data structure **114** is in the general form of a list of Node objects (one per URL) and Edge objects (one per link), with associated information attached as attributes of these objects. For each URL of the site, the information stored within the Site Graph typically includes the following: the URL type, the scanning status (OK, not found, inaccessible, unread, or access denied), the data and time of last modification, the URLs (addresses) of all incoming and outgoing links, the file size (if the URL was actually retrieved), an annotation, and the associated protocol.

Periodically during the scanning process, the main thread **120** executes a Visual Web Display routine which applies the Solar Layout method to the URLs and links of the Site Graph **114**. (The term "routine," as used herein, refers to a functionally-distinguishable portion of the executable code of a larger program or software package, but is not intended to imply the modularity or callability of such code portion.) As indicated above, this method selects the links to be displayed within the site map (by selecting a span tree from the graph structure), and determines the layout and size for the display of the nodes (URLs) and non-hidden links of the map. The execution of this display routine results in modifications to the display attributes of the nodes (Node objects) and links (Edge objects) of the Site Graph **114** in accordance with the above-described layout and display principles. For example, for each link which is not present in the span tree, the visibility attribute of the link is set to "hidden." (As described below, link and node attributes are also modified in response to various user actions during the viewing of the map, such as the application of filters to the site map.)

In the preferred embodiment, the Visual Web Display routine is executed each time a predetermined threshold of new URLs have been scanned. Each time the routine is executed, the screen is automatically updated (in Visual Web Display format) to show the additional URLs that have been identified since the last execution of the routine. This allows the user to view the step-by-step generation of the site map during the scanning process. The user can selectively pause and restart the scanning process using respective controls on the Astra toolbar **46**.

FIG. **10** illustrates the general decision process followed by a scanning thread **122** when a URL is scanned. This process implements the above-mentioned caching scheme for reducing unnecessary downloads of URLs and URL headers during Automatic Update operations. With reference to decision block **140**, it is initially determined whether the URL has previously been scanned. If it has not been scanned, the thread either requests the file from the server (if the URL is an HTML file), or else requests the URL's header from the server, as illustrated by blocks **142–146**. (URL headers are retrieved using the HEAD method of the HTTP protocol.) In either case, the scanning thread waits for the server to respond, and generates an appropriate status code (such as a code indicating that the URL was not found or was inaccessible) if a timeout occurs or if the server returns an error code, as indicated by block **150**.

If, on the other hand, the URL has previously been mapped (block **140**), the date/time of last modification stored in the Site Graph **114** (FIG. **9**) is used to determine whether or not a retrieval of the URL is necessary. This is accomplished using standard argument fields of the HTTP "GET" method which enable the client to specify a "date/time of last modification" condition for the return of the file. With reference to blocks **158** and **160**, the GET request is for the entire URL if the file is an HTML file (block **158**), and is for the URL header if the file is a non-HTML file (block **160**). Referring again to block **150**, the thread then waits for the server response, and returns an appropriate status code if an error occurs.

As indicated by block **164**, if an HTML file is returned as the result of the server request, the scanning thread parses the HTML and identifies any links within the file to other URLs. As indicated above, the main thread **120** launches additional scanning threads **122** to scan these URLs if any links are detected, with the exception that external links are not scanned unless a "verify external links" option has been selected by the user.

As indicated by the foregoing, the scanning process of the present invention provides a high degree of bandwidth efficiency by avoiding unnecessary retrievals of URLs and URL headers that have not been modified since the previous mapping, and by using multiple threads to scan the Web site.

VII. Scanning and Mapping of Dynamically-Generated Pages (FIGS. **11–15**)

A feature of the invention which permits the scanning and mapping of dynamically-generated Web pages will now be described. By way of background, a dynamically-generated Web page ("dynamic page") is a page that is generated "on-the-fly" by a Web site in response to some user input, such as a database query. Under existing Web technology, the user manually types-in the information (referred to herein as the "dataset") into an embedded form of an HTML document while viewing the document with a Web browser, and then selects a "submit" type button to submit the dataset to a Web site that has back-end database access or real-time data generation capabilities. (Technologies which provide such Web server extension capabilities include CGI, Microsoft's ISAPI, and Netscape's NSAPI.) A Web server extension module (such as a CGI script) then processes the dataset (by, for example, performing a database search, or generating real-time data) to generate the data to be returned to the user, and the data is returned to the browser in the form of a standard Web page.

One deficiency in existing Web site mapping programs is that they do not support the automatic retrieval of dynamic pages. As a result, these mapping programs are not well suited for tracking changes to back-end databases, and do not provide an efficient mechanism for testing the functionality of back-end database search components. In accordance with one aspect of the invention, these deficiencies are overcome by providing a mechanism for capturing datasets entered by the user into a standard Web browser, and for automatically re-submitting such datasets during the updating of site maps. The feature of Astra which provides these capabilities is referred to as Dynamic Scan.

FIG. **11** illustrates the general flow of information between components during a Dynamic Scan capture session, which can be initiated by the user from the Astra tool bar. Depicted in the drawing is a client computer **92** communicating with a Web site **113** over the Internet **110** via respective TCP/IP layers **108**, **178**. The Web site **113** includes a Web server application **112** which interoperates with CGI scripts (shown as layer **180**) to generate Web pages on-the-fly. Running on the client computer **92** in conjunction with the Astra application **94** is a standard Web browser **170** (such as Netscape Navigator or Microsoft's Internet Explorer), which is automatically launched by Astra when the user activates the capture session. As illustrated, the Web browser **170** is configured to use the Astra application **94** as an HTTP-level proxy. Thus, all HTTP-level messages (client requests) generated by the Web browser **170** are initially passed to Astra **94**, which in-turn makes the client requests on behalf of the Web browser. Server responses (HTML pages, etc.) to such requests are returned to Astra by the client computer's TCP/IP layer **108**, and are then forwarded to the browser to maintain the impression of normal browsing.

During the Dynamic Scan capture session, the user types-in data into one or more fields **174** of an HTML document **172** while viewing the document with the browser **170**. The HTML document **172** may, for example, be an internal URL which is part of a Web site map, or may be an external URL which has been linked to the site map for mapping purposes. When the user submits the form, Astra extracts the manually-entered dataset, and stores this dataset (in association with the HTML document **172**) for subsequent use. When Astra subsequently re-scans the HTML document **172** (during an Automatic Update of the associated site map), Astra automatically retrieves the dataset, and submits the dataset to the Web site **113** to recreate the form submission. Thus, for example, once the user has typed-in and submitted a database query in connection with a URL of a site map, Astra will automatically perform the database query (and map the results, as described below) the next time an Automatic Update of the map is performed.

With further reference to FIG. **11**, when the Web site **113** returns the dynamic page during the capture session (or during a subsequent Automatic Update session), Astra automatically adds a corresponding node to the site map, with this node being displayed as being linked to the form page. (Screen displays taken during a sample capture session are shown in FIGS. **13–15** and are described below.) In addition, Astra parses the dynamic page, and adds respective nodes to the map for each outgoing link of the dynamic page. (In the default setting, these outgoing links are not scanned.) Astra also parses any static Web pages that are retrieved with the browser during the Dynamic Scan capture session, and updates the site map (by appending appropriate URL icons) to reflect the static pages.

FIG. **12** illustrates the general flow of information during a Dynamic Scan capture session, and will be used to describe the process in. greater detail. Labeled arrows in FIG. **12** represent the flow of information between software and database components of the client and server computers. As will be apparent, certain operations (such as updates to the map structure **128**) need not be performed in the order shown.

Prior to initiating the Dynamic Scan session, the user specifies a page **172** which includes an embedded form. (This step is not shown in FIG. **12**). This can be done by browsing the site map with the Astra GUI to locate the node of a form page **172** (depicted by Astra using a special icon), and then selecting the node with the mouse. The user then initiates a Dynamic Scan session, which causes the following dialog to appear on the screen: YOU ARE ABOUT TO ENTER DYNAMIC SCAN MODE. IN THIS MODE YOU WORK WITH A BROWSER AS USUAL, BUT ALL YOUR ACTIONS (INCLUDING FORM SUBMISSIONS) ARE RECORDED IN THE SITE MAP. TO EXIT FROM THIS MODE, PRESS THE "STOP DYNAMIC SCAN" BUTTON ON THE MAIN TOOLBAR OR CHOOSE THE "STOP DYNAMIC SCAN" OPTION IN THE SCAN MENU.

When the user clicks on the "OK" button, Astra modifies the configuration of the Web browser **170** within the registry **182** of the client computer to set Astra **94** as a proxy of the browser, as illustrated by arrow A of FIG. **12**. (As will be recognized by those skilled in the art, the specific modification which needs to be made to the registry **182** depends upon the default browser installed on the client computer.) Astra then launches the browser **170**, and passes the URL (address) of the selected form page to the browser for display. Once the browser has been launched, Astra modifies the registry **182** (arrow B) to restore the original browser

configuration. This ensures that the browser will not attempt to use Astra as a proxy on subsequent browser launches, but does disable the browser's use of Astra as a proxy during the Dynamic Scan session.

As depicted in FIG. **12**, the browser **170** retrieves and displays the form page **172**, enabling the user to complete the form. In response to the submission by the user of the form, the browser **170** passes an HTTP-level (GET or POST) message to Astra **94**, as indicated by arrow C. This message includes the dataset entered by the user, and specifies the URL (address) of the CGI script or other Web server extension component **180** to which the form is addressed. Upon receiving this HTTP message, Astra displays the dialog "YOU ARE ABOUT TO ADD A DATA SET TO THE CURRENT URL IN THE SITE MAP," and presents the user with an "OK" button and a "CANCEL" button.

Assuming the user selects the OK button, Astra extracts the dataset entered by the user and then forwards the HTTP-level message to its destination, as illustrated by arrow E. In addition, as depicted by arrow D, Astra stores this dataset in the Site Graph **114** in association with the form page **172**. As described above, this dataset will automatically be retrieved and re-submitted each time the form page **172** is re-scanned as part of an Automatic Update operation. With reference to arrows F and G, when the Web server **112** returns the dynamic page **184**, Astra **94** parses the page and updates the Site Graph **114** to reflect the page and any outgoing links of the dynamic page. (In this regard, Astra handles the dynamic page in the same manner as for other HTML documents retrieved during the normal scanning process.) In addition, as depicted by arrow H, Astra forwards the dynamic page **184** to the Web browser **170** (which in-turn displays the page) to maintain an impression of normal Web browsing.

Following the above sequence, the user can select the "stop dynamic scan" button or menu option to end the capture session and close the browser **170**. Alternatively, the user can continue the browsing session and make additional updates to the site map. For example, the user can select the "back" button **186** (FIG. 14) of the browser to go back to the form page and submit a new dataset, in which case Astra will record the dataset and resulting page in the same manner as described above.

Although the system of the preferred embodiment utilizes conventional proxy technology to redirect and monitor the output of the Web browser **170**, it will be recognized that other technologies and redirection methods can be used for this purpose. For example, the output of the Web browser could be monitored using conventional Internet firewall technologies.

FIGS. **13–15** are a sequence of screen displays taken during a Dynamic Scan capture session in which a simple database query was entered into a search page of the Infoseek™ search engine. FIG. **13**, which is the first display screen of the sequence, illustrates a simple map **190** generated by opening a new map and then specifying http://www.infoseek.com/ as the URL. Displayed at the center of the map is the form page icon for the Infoseek™ search page. The 20 children **192** of the form page icon correspond to external links (i.e., links to URLs outside the infoseek-.com domain), and are therefore displayed using the "not scanned" icon. (As described above, if the "verify external links" option of Astra is selected, Astra will verify the presence of such external URLs and update the map accordingly.)

FIG. **14** illustrates a subsequent screen display generated by starting a Dynamic Scan session with the Infoseek™ page selected, and then typing in the word "school" into the query field **194** of the page. (Intermediate displays generated by Astra during the Dynamic Scan session are omitted.) As illustrated in the figure, the Web browser comes up within a window **196**, allowing the user to access the Astra controls and view the site map **190** during the Dynamic Scan session.

FIG. **15** illustrates the updated map **190'** generated by Astra as a result of the FIG. **14** database query. The node (icon) **200** labeled "titles" in the map represents the dynamic page returned by the Infoseek™ Web site, and is depicted by Astra as being linked to the Infoseek™ form page. A special "dynamic page" icon **200** is used to represent this newly-added node, so that the user can readily distinguish the node from nodes representing statically-generated pages. The children **204** of the dynamic page node **200** represent outgoing links from the dynamic page, and are detected by Astra by parsing the HTML of the dynamic page. In the present example, at least some of the children **204** represent search results returned by the Infoseek search engine and listed in the dynamic page.

As generally illustrated by FIG. **15**, in which the children **204** of the dynamic page **200** are represented with Astra's "not scanned," Astra does not automatically scan the children of the dynamically-generated Web page during the Dynamic Scan session. To effectively scan a child page **204**, the user can retrieve the page with the browser during the Dynamic Scan session, which will cause Astra to parse the child page and update the map accordingly.

Following the sequence illustrated by FIGS. **13–15**, the user can, for example, save the map **190'** to disk, which will cause the corresponding Site Graph **114** to be written to disk. If the user subsequently retrieves the map **190'** and initiates an Automatic Update operation, Astra will automatically submit the query "school" to the Infoseek™ search engine, and update the map **190'** to reflect the search results returned. (Children **204** which do not come up in this later search will not be displayed in the updated map.) By comparing this updated map to the original map **190'** (either manually or using Astra's map comparison tool), the user can readily identify any new search result URLs that were returned by the search engine.

While the above-described Dynamic Scan feature is particularly useful in Web site mapping applications, it will be recognized that the feature can also be used to in other types of applications. For example, the feature can be used to permit the scanning of dynamically-generated pages by general purpose Webcrawlers. In addition, although the feature is implemented in the preferred embodiment such that the user can use a standard, stand-alone Web browser, it will be readily apparent that the feature can be implemented using a special "built-in" Web browser that is integrated with the scanning and mapping code.

VIII. Display of Filtered Maps (FIGS. **16–18**)

The content, status and location filters of Astra provide a simple mechanism for allowing the user to focus-in on URLs which exhibit particular characteristics, while making use of the intuitive layout and display methods used by Astra for the display of site maps. To apply a filter, the user simply selects the corresponding filter button on the filter toolbar **47** while viewing a site map. (The specific filters that are available within Astra are listed above under the heading ASTRA GRAPHICAL USER INTERFACE.) Astra then automatically generates and displays a filtered version of the map. In addition to navigating the filtered map using Astra's navigation controls, the user can select the Visual Web Display button **73** (FIG. **16**) to view the filtered map in Astra's VWD format. Combinations of the filters can be applied to the site map concurrently.

FIG. **16** illustrates the general display format used by Astra when a filter is initially applied to a site map. This example was generated by selecting the "hide OK URLs" button **220** on the filter toolbar **47** while viewing a site map similar to the map **30** of FIG. **1**. As illustrated by the screen display, the selection of the filter causes Astra to generate a filtered map **30'** which is in the form of skeletal view of the original map, with only the links and URLs of interest remaining.

As generally illustrated by FIG. **16**, the filtered map **30'** consists primarily of the following components of the original map **30**: (i) the URLs which satisfy (pass through) the filter, (ii) the links to the URLs which satisfy the filter, and (iii) all "intermediate" nodes and links (if any) needed to maintain connectivity between the root (home page) URL and the URLs which satisfy the filter. (This display methodology is used for all of the filters of the filter toolbar **47**, and is also used when multiple filters are applied.) In this example, the filtered map **30'** thus consists of the home page. URL, all URLs which have a scanning status other than "OK," and the links and nodes needed to maintain connectivity to the non-OK URLs. To allow the user to readily distinguish between the two types of URLs, Astra displays the URLs which satisfy the filter in a prominent color (such as red) when the filtered map is viewed in a zoomed-out mode. The general process used by Astra to generate the skeletal view of the filtered map is illustrated by FIG. **17**.

While viewing the filtered map, the user can perform any of a number of actions, such as zoom in and out to reveal additional URL information, launch editor programs to edit the displayed URLs, and apply additional filters to the map. In addition, the user can select the Visual Web Display button **73** to display the filtered map in Astra's VWD format. To restore the hidden nodes and links to the map, the user clicks on the selected filter button to remove the filter.

FIG. **18** illustrates the filtered map of FIG. **16** following selection by the user of the VWD button **73**. As generally illustrated by these two figures, the selection of the VWD button **73** causes Astra to apply the Solar Layout method to the nodes and links of the filtered map. In addition, to provide the user with a contextual setting for viewing the remaining URLs, Astra restores the visibility of selected nodes and links in the immediate vicinity of the URLs that satisfy the filter. As generally illustrated by node icons **226**, **228** and **230** in FIG. **18**, an icon color coding scheme is used to allow the user to distinguish the URL icons which satisfy the filter from those which do not, and to allow the user to distinguish URLs which have not been scanned.

LX. Tracking and Display of Visitor Activity (FIGS. **19** and **20**)

An important feature of Astra is its the ability to track user (visitor) activity and behavior patterns with respect to a Web site and to graphically display this information (using color coding, annotations, etc.) on the site map. In the preferred embodiment, this feature is implemented in-part by the Action Tracker plug-in, which gathers user activity data by retrieving and analyzing server log files commonly maintained by Web servers. Using this feature, Webmasters can view site maps which graphically display such information as: the most frequently-accessed URLs, the most heavily traveled links and paths, and the most popular site entry and exit points. As will be appreciated by those skilled in the art, the ability to view such information in the context of a site map greatly simplifies the task of evaluating and maintaining Web site effectiveness.

By way of background, standard Web servers commonly maintain server access log files ("log files") which include information about accesses to the Web site by users. These files are typically maintained in one of two standard formats: the HTTP Server Access Log File format, or the HTTP Server Referrer Log File format. (Both of these formats are commonly used by Web servers available from Microsoft, Netscape, and NSCA, and both formats are supported by Astra.) Each entry (line) in a log file represents a successful access to the associated Web site, and contains various information about the access event. This information normally includes: the path to the accessed URL, an identifier of the user (typically in the form of an IP address), and the date and time of the access. Each log file stored on a physical server typically represents some window of time, such as one month.

In accordance with the invention, Astra uses the information contained within a log file in combination with the associated site graph to determine probable paths taken by visitors to the Web site. (The term "visitor" is used herein to distinguish the user of the Web site from the user of Astra, but is not intended to imply that the Web site user must be located remotely from the Web site.) This generally involves using access date/time stamps to determine the chronological sequence of URLs followed by each visitor (on a visitor-by-visitor basis), and comparing this information against link information stored in the site map (i.e., the Site Graph object **114**) to determine the probable navigation path taken between the accessed URLs. (This method is described in more detail below.) By determining the navigation path followed by a visitor, Astra also determines the site entry and exits points taken by the visitor and all of the links traversed by the visitor. By performing this method for each visitor represented in the log file and appropriately combining the information of all of the visitors, Astra generates statistical data (such as the number of "hits" or the number of exit events) with respect to each link and node of the Web site, and attaches this information to the corresponding Node and Edge objects **115**, **116** (FIG. **8**) of the Site Graph **114**.

To activate the Action Tracker feature, the user selects the Action Tracker option from the TOOLS menu while viewing a site map. The user is then presented with the option of either retrieving the server log file or loading a previously-saved Astra Activity File. Astra Activity Files are compressed versions of the log files generated by Astra and stored locally on the client machine, and can be generated and saved via controls within the Action Tracker plug-in. Astra also provides an option which allows the user to append a log file to an existing Astra Activity file, so that multiple log files can be conveniently combined for analysis purposes. Once the Activity File or server log file has been loaded, an Action Tracker dialog box (FIG. **19**) opens which provides controls for allowing the user to selectively display different types of activity data on the map. In one implementation (described separately below under the heading AUTOMATED GENERATION OF LOAD TESTING SCENARIOS), the Action Tracker dialog box includes a "Load Wizard" button for allowing the user to initiate the automatic generation of a load-testing scenario from the activity data.

FIG. **19** illustrates the general display format used by the Action Tracker plug-in to display activity levels on the links of a site. As illustrated by the screen display, the links between URLs are displayed using a color-coding scheme which allows the user to associate different link colors (and URL icon colors) with different relative levels of user activity. As generally illustrated by the color legend, three distinct colors are used to represent three (respective) adjacent ranges of user activity.

In the illustrated display mode (uncolored links hidden, uncolored URLs not hidden), all of the URLs of the site map are displayed, but the only links that are displayed are those which satisfy a user-adjustable minimum activity threshold. Each visible link is displayed as a one-way arrow (indicating the link direction), and includes a numerical annotation indicating the total number of hits revealed by the log or activity file. The number of hits per URL can be viewed in List View mode in a corresponding column. As can be seen from an observation of the screen display, the displayed links include links which do not appear in the Visual Web Display view of the map.

With further reference to FIG. **19**, a slide control **240** allows the user to adjust the "hits" thresholds corresponding to each of the three colors. By clicking and dragging the slide control, the user can vary the number of displayed links in a controllable manner to reveal different levels of user (visitor) activity. This feature is particularly useful for identifying congested links, which can be remedied by the addition of appropriate data redundancies.

FIG. **20** illustrates the general process used by the Action Tracker plug-in to detect the link activity data (number of hits per link) from the log file. The displayed flow chart assumes that the log file has already been retrieved, and that the attribute "hits" has been defined for each link (Edge object) of the Site Graph and set to zero. As illustrated by the flow chart, the general decision process is applied line-by-line to the log file (each line representing an access to a URL) until all of the lines have been processed. With reference to blocks **250** and **252**, each time a new line of the log file is ready, it is initially determined whether or not the log file reflects a previous access by the user to the Web site. This determination is made by searching for other entries within the log file which have the same user identifier (e.g., IP address) and an earlier date/time stamp.

Blocks **254** and **256** illustrate the steps that are performed if the user (visitor) previously visited the site. Initially, the Site Graph is accessed to determine whether a link exists from the most-recently accessed URL to the current URL, as indicated by decision block **254**. If such a link exists, it is assumed that the visitor used this link to get to the current URL, and the usage level ("hits" attribute) of the identified link is incremented by one. If no such link is identified between the most-recently accessed URL and the current URL, an assumption is made that the user back-tracked along the navigation path (by using the "BACK" button of the browser) before jumping to the current URL. Thus, decision step **254** is repeated for each prior access by the user to the site, in reverse chronological order, until either a link to the current URL is identified or all of the prior accesses are evaluated. If a link is detected during this process, the "hits" attribute of the link is incremented.

As indicated by block **258**, the above process continues on a line-by-line basis until all of the lines of the log file have been processed. Following the execution of this routine, the "hits" attribute of each link represents an approximation (based on the above assumptions) of the number of times the link was traversed during the time frame represented by the log file.

As will be apparent, the general methodology illustrated by the FIG. **20** flow chart can be used to detect a variety of different types of activity information, which can be superimposed on the site map (by modifying node and link display attributes) in the same general manner as described above. The following are examples of some of the types of activity data that can be displayed, together with descriptions of several features of the invention which relate to the display of the activity data:

Exit Points. Exit points are deduced from the log file on a visitor-by-visitor basis by looking for the last URL accessed by each visitor, and by looking for large time gaps between consecutive accesses to the site. An "exits" attribute is defined for each node. to keep track of the total number of exit events from each node. The color-coding scheme described above is then used to allow the user to controllably display different thresholds of exit events.

Usage Zones. When viewing a large site map in its entirety (as in FIG. **1**), it tends to be difficult to identify individual URL icons within the map. This in-turn makes it difficult to view the color-coding scheme used by the Action Tracker plug-in to display URL usage levels. The Usage Zones feature alleviates this problem by enlarging the size of the colored URL icons (i.e., the icons of nodes which fall within the predetermined activity level thresholds) to a predetermined minimum size. (This is accomplished by increasing the "display size" attributes of these icons.) If these colored nodes are close together on the map, the enlarged icons merge to form a colored zone on the map. This facilitates the visual identification of high-activity zones of the site.

Complete Path Display. With this feature, the complete path of each visitor is displayed on the map on a visitor-by-visitor basis, with the visitor identifier and the URL access time tags displayed in the List View window **78** (FIG. **4**). This feature permits fine-grain inspection of the site usage data, which is useful, for example, for analyzing security attacks and studying visitor behavior patterns.

Log Filters. Because server access log files tend to be large, it is desirable to be able to filter the log file and to display only certain types of information. This feature allows the user to specify custom filters to be applied to the log file for purposes of limiting the scope of the usage analysis. Using this feature, the user can, for example, specify specific time and date ranges to monitor, or limit the usage analysis to specific IP addresses or domains. In addition, the user can specify a minimum visit duration which must be satisfied before the Action Tracker will count an access as a visit.

X. Map Comparison Tool (FIG. **21**)

FIG. **21** illustrates a screen display generated using Astra's Change Viewer map comparison tool. As illustrated by the screen display, the comparison tool generates a comparison map **268** which uses a color-coding scheme to highlight differences between two site maps, allowing the user to visualize the changes that have been made to a Web site since a prior mapping of the site. Using the check boxes within the Change Viewer dialog box **270**, the user can selectively display the following: new URLs and links, modified URLs, deleted URLs and links, and unmodified URLs and links. As illustrated, each node and link of the comparison map is displayed in one of four distinct colors to indicate its respective comparison status: new, modified, deleted, or unmodified.

To compare two maps, the user selects the "Compare Maps" option from the TOOLS menu while viewing the current map, and then specifies the filename of the prior map. Astra then performs a node-by-node and link-by-link comparison of the two map structures (Site Graphs) to identify the changes. This involves comparing the "URL" attributes of the associated Node and Edge objects to identify URLs and links that have been added and deleted, and comparing the "date/time of last modification" attributes of

like Node objects (i.e., Node objects with the same "URL" attribute) to identify URLs that have been modified. During this process, a comparison map data structure is generated which reflects the comparison of the two maps, using color attributes to indicate the comparison outcomes (new, modified, deleted or unmodified) of the resulting nodes and links. Once the comparison map data structure has been generated, Astra applies the Solar Layout method to the structure and displays the comparison map **268** in Astra's VWD format. (The user can also view the comparison map in Astra's "incoming links" and "outgoing links" display modes.) The user can then adjust the "show" settings in the dialog box **270**, which causes Astra to traverse the comparison map data structure and adjust the visibility attributes according to the current settings.

In one embodiment, the Astra code includes an Automated Comparison feature which allows the user to schedule the automatic generation of comparison maps on a periodic basis (e.g., once a week). When this feature is used, Astra automatically re-scans the site at the scheduled times, and then uses the superseded and the new map files to generate and save a comparison map. One variation of this feature involves automatically sending an email containing a list of the new and modified URLs (and possibly other comparison data) to a pre-specified individual whenever the automatic comparison takes place. The email (and/or the comparison map) can then be used by the individual, for example, to efficiently review the additions to the Web site.

XI. Link Repair Plug-in (FIG. **22**)

FIG. **22** illustrates the operation of Astra's Link Doctor plug-in. To access this feature, the user selects the "Link Doctor" option from the TOOLS menu while viewing a site map. The Link Doctor dialog box **284** then appears with a listing (in the "broken links" pane **286**) of all of the broken links (i.e., URLs of missing content objects) detected within the site map. (Astra detects the missing links by searching the Site Graph for Node objects having a status of "not found.") When the user selects a URL from the broken links pane (as illustrated in the screen display), Astra automatically lists all of the URLs which reference the missing content object in the "appearing in" pane **288**. This allows the user to rapidly identify all of the URLs (content objects) that are directly affected by the broken link.

In addition to listing all of the referencing URLs in the "appearing in" pane **288**, Astra generates a graphical display (in Astra's "incoming links" display mode) which shows the selected (missing) URL **290** and all of the URLs **292** which have links to the missing URL. In this example, the missing URL is a GIF file which is embedded within eight different HTML files **292**. From the display shown in FIG. **22**, the user can select one of the referencing nodes **292** (by either clicking on its icon or its listing in the "appearing in" pane), and then select the "Edit" button **296** to edit the HTML document and eliminate the reference to the missing file.

XII. Automated Generation of Load Testing Scenarios (FIGS. **25–32**)

An important feature of the invention involves the ability to automatically generate load testing scripts, and associated "scenario files," from server access log files of Web sites. In the preferred embodiment, this feature (referred to as "Load Wizard") is implemented within a modified version of the above-described Action Tracker plug-in, and is invoked by selecting a Load Wizard button **300** (FIG. **27**) of the Action Tracker dialog box. The test scripts and scenario files generated via the Load Wizard feature can be used to load-test a Web site using either the LoadRunner product or the Astra SiteTest product of Mercury Interactive. Although this feature is described with reference to the load-testing of Web sites, the scenario generation methodology can be applied to the testing of other types of server applications that maintain access logs. For instance, the methodology could be used to test a mainframe terminal application for which a central machine maintains a log of accesses to different screens.

To facilitate an understanding of the Load Wizard feature, an overview is initially provided of the Web site load-testing features of the commercially-available LoadRunner and Astra SiteTest products, including the scenario generation features of these products. Additional information about these features, and about the LoadRunner and Astra SiteTest products generally, is available from Mercury Interactive Corporation of Sunnyvale California. As in the above description of the Action Tracker plug-in, the term "user" is used in the following Load Wizard description to refer to users of the disclosed Web site testing tools (e.g., Astra and Astra SiteTest), and the term "visitor" is used to refer to users of the subject Web site.

(a) Web Site Testing with LoadRunner and SiteTest

As described above, LoadRunner and Astra SiteTest Products (hereinafter "LoadRunner" and "SiteTest") use pre-recorded load testing scripts, referred to herein as "Web scripts," to conduct load tests of Web sites. Each Web script consists essentially of a sequence of HTTP messages (stored within a script file), with each message representing a client request to a Web site. The following is an example of a Web script consisting of three URL request messages:

```
URL("http://www.merc-int.com/forms/edu_reg.html");
URL("http://www.merc-int.com/cgi-bi/index.html");
URL("http://www.merc-int.com/cgi-bi/login.pl");
```

Form submissions and other types of requests that invoke "back end" Web site components can be included.

In current implementations of LoadRunner and SiteTest, the Web scripts are generated by capturing and recording the output of a standard Web browser during interactive browsing of the site by a user. The browser output is captured and recorded using a "Web Vuser Generator" component, which uses the proxy-based capture technique (illustrated in FIGS. **11** and **12** and described above) used with the Dynamic Scan feature. The Web scripts can also be typed-in and/or edited manually.

During the load testing process, Web scripts are sequentially played back or "run" by sequentially submitting the request messages to the site. This is preferably accomplished using a Vuser executable. As depicted in FIG. **25**, multiple Vusers (i.e., multiple instances of the Vuser executable) can be run simultaneously on a single workstation, with different Vusers optionally running different Web scripts. This produces a load in which multiple client requests can (and typically will) be pending at-a-time, as is commonly the case when large numbers of concurrent visitors are accessing the site. As illustrated in FIG. **25**, the Vusers communicate with and run under the control of the LoadRunner or SiteTest Controller **298** (both referred to herein simply as "the Controller"). As illustrated by the partial screen display of FIG. **26**, the Controller **298** includes a user interface that allows the user to selectively launch and monitor the Vusers.

During the load testing process, each Vuser monitors the Web site's responses to the client requests submitted by that Vuser, and records various performance-related characteristics of these responses. These characteristics include, for example, response times to individual client requests, timeout events, and error events. Following the load testing

process, the user is presented with a set of graphical reports that allow the user evaluate the site's performance.. Using these reports, the user can, for example, compare response times of different site components (Web servers, CGI scripts, APIs, proxy servers, etc.) to identify bottlenecks and other performance problems.

To facilitate the formulation of repeatable, multi-Vuser load tests, LoadRunner and SiteTest include code for allowing the user to define a test "scenario" that specifies the details of the test. A scenario may, for example, represent ten users that are concurrently attempting to access a particular back-end database of a Web site. Within a scenario, Vusers can be arranged into groups (referred to as "Simulation Groups" or "Sgroups") that run the same Web script. For example, an Sgroup of 10 Vusers can be formed to run a Web script "register.txt," and an Sgroup of 5 Vusers can be formed to run a Web script "browse.txt." (The screen display of FIG. **26** illustrates an Sgroup "G1" consisting of three Vusers that run the Web script "WEBSITES.TXT.") Once a scenario has been defined, the scenario can be loaded and run repeatedly via the Controller **298**.

To define a scenario, the user initially uses the Web Vuser Generator component (not shown) to generate the Web scripts to be included within the scenario (assuming these Web scripts do not already exist). The user then invokes a Scenario Wizard component (not shown) to formulate the scenario. Using the Scenario Wizard, the user specifies such details as the number of Vusers, the Web script (identified by file name) to be run by each Vuser, and the number of consecutive times ("loops") that each Web script is to be run during scenario execution. The user can also define one or more Sgroups, and can specify various testing parameters (such a transaction timeout period) to be used during execution of the scenario. Once the scenario has been defined, the details of the scenario are stored in a scenario file under a user-specified filename. The scenario file and the associated script file(s) fully define the scenario.

(b) Overview of Scenario Generation Process

An objective of the present invention is to reduce the complexity of the above-described scenario generation process, including the process of generating Web scripts. Another objective is to provide a mechanism for generating testing scenarios that more accurately represent load distributions present during typical site usage.

In accordance with these objectives, a code module is provided that automatically generates a load testing scenario (including the scenario file and the Web script files) from information stored within server access log files. Because the process is automatic, the user does not have to record the output of the browser, define Sgroups, and perform other activities normally associated with scenario generation.

An important benefit of the process is that the resulting scenarios closely represent the load distributions reflected by the server access log files. Thus, for example, if a log file indicates heavy access to a first area (e.g., URL) of a site and relatively light access to a second area of the site, the resulting scenario will stress the first area more heavily than the second area (according to the general proportions indicated by the log file). Because the log files represent the actual browsing behaviors of past visitors to the site (and typically large cross sections of visitors), the load tests accurately emulate realistic load conditions.

FIGS. **27–29** are partial screen displays which illustrate, in example form, the sequence of events (as seen by the user) that occur when the user invokes the Load Wizard feature to generate a scenario. With reference to FIG. **27**, the user initially loads-in a log file (or possibly multiple log files) while viewing a map of the site. In one implementation, the log file may be either an HTTP Server Access Log File or an HTTP Server Referrer Log File. Alternatively, the user may load-in an Astra Activity File that was previously saved. As described above, an Astra Activity File is a condensed file format used by the Action Tracker plug-in to store server access log files.

Once the log file has been loaded-in, the associated site usage information is displayed on the map according to the current Action Tracker settings, as illustrated in FIG. **27** and described above. At this point, the user can select the Load Wizard button **300** from the Action Tracker dialog box to bring up the "create site test scenario" dialog box of FIG. **28**. From this dialog box, the user can enter the name of the scenario to be generated, and can specify the number of Vusers (up to 50) and the number of loops. The number of Vusers controls the magnitude of the load to be applied to the site. The number of loops controls the duration of the test. In this example, the user has specified a scenario name of "TEST5, " and has selected test parameters of thirty Vusers and ten loops.

Once the user selects the "OK" button **304**, the Load Wizard code implements a two-phase process (discussed below) to generate the load testing scenario, which, as indicated above, is represented as a scenario file and at least one script file. As illustrated in FIG. **29**, a dialog box **310** is then displayed to the user indicating the results of the automated scenario generation process. In this example, the dialog message indicates that the scenario "TEST5" was successfully generated, and that execution of the scenario will produce 22,200 hits to the Web server. From this dialog box **310**, the user can select the "YES" button to launch the Controller and run the scenario.

FIG. **30** is a flow diagram which illustrates the general flow of information during the automated scenario generation process. The entity that generates the scenario is represented in this figure as the Load Wizard module **320**. The inputs to the Load Wizard module **320** are a list of visitor routes (extracted from the server log access file), the user-specified numbers of Vusers and loops, and the scenario name. The number of loops and the scenario name are shown in parentheses to indicate that they are not part of the two-phase process used to reduce server log file information to Web scripts.

Each route within the routes list is in the general form:

URL1-URL2 (occurrence),

wherein "URL1-URL2" represents a link from URL1 to URL2, and "occurrences" is a numerical value representing the number of times this route was taken by a visitor (as represented within the server access log file). The general process used to translate the server log file data into this format is illustrated in FIG. **20** and described above. In one implementation, the FIG. **20** process is implemented such that all non-HTML URLs are filtered out, leaving only the routes between Web documents. As illustrated in the example that follows, the scenario generation process implemented by the Load Wizard module **320** can handle routes that include three or more URLs.

With further reference to FIG. **30**, the outputs of Load Wizard module **320** include one or more script files **322** (each containing a respective Web script, in textual form), and a scenario file **324**. The scenario file **324** is a text file that includes various control information of the scenario, including the filenames and paths of the script files, the allocations

of Vusers to script files (including any Sgroup definitions), and the number of loops for running the scripts. The Web scripts and other scenario information can, of course, be stored in other forms. For example, the scenario information can be stored as one or more executable files that run during the load testing process.

As depicted in FIG. **30**, the Load Wizard module **320** applies a two-phase process to the input data to generate the scenario. This process is illustrated in FIGS. **31** and **32**, and is described below. Following the generation of the scenario, the Load Wizard module **320** launches the SiteTest or Loadrunner Controller **328** (assuming the user selects the "YES" option in the FIG. **29** dialog). Upon launching the Controller **328**, the Load Wizard module **320** passes to the Controller the paths and filenames of the scenario and Web script files to allow the Controller to bring-up the scenario for execution. In one embodiment, the Web scripts and other scenario data are automatically compiled into an executable form prior to the running of the scenario.

(c) Two-Phase Translation Process

FIGS. **31** and **32** illustrate phases 1 and 2, respectively, of the translation process implemented by the Load Wizard module **320** to generate a scenario. The description of the process will be provided in conjunction with a simple example in which the number of Vusers specified by the user is ten, and in which the routes list is as follows:

A-B-C (100)
B-D (200)
A-B (250)
C-F (80)

In this hypothetical routes list, each alphabetic character (A–F) represents a unique URL of the Web site to be tested, and the number in parentheses is the "occurrences" value of the corresponding route.

The general goal of the two-phase process is to translate the routes list into a set of Web scripts to be run by the Vusers, while preserving the general load distribution represented by the routes list. In performing this task, the process seeks to generate Web scripts that are approximately equal in length, so that each Vuser will generate roughly the same number of hits during scenario execution.

With reference to FIG. **31**, the first phase of the process attempts to merge consecutive routes while maintaining the load distribution represented by the routes list. As represented by decision block **330**, a determination is initially made whether the sum of the occurrences (100+200+250+80=630 in this example) is greater than the number of Vusers (ten in this example). On the first pass of the phase 1 program loop, the sum of the occurrences will normally greatly exceed the number of Vusers, since the log file will typically represent all hits to the site over an extended period of time (e.g., one week). If the sum of the occurrences exceeds the number of Vusers, a search is conducted of the routes list for a pair of consecutive routes (i.e., a pair of routes R1, R2 for which the last URL of R1 matches the first URL of R2), as indicated by decision block **332**. If no such pair exists within the routes list, or if the sum of the occurrences does not exceed the number of Vusers, the process proceeds to phase **2** (FIG. **32**), as indicated by block **334**.

With reference to blocks **338** and **340**, if a pair of consecutive routes (R1, R2) is found, R1 and R2 are merged to form a new route which replaces the route (R1 or R2) having the smaller occurrence value. The new route is assigned the lower of the occurrence values of R1 and R2, and the remaining (unreplaced) route of R1 and R2 is assigned an occurrence value equal to the difference between the occurrences of R1 and R2. In this example, the routes A-B-C (100) and C-F (80) are merged to form a new route A-B-C-F, which replaces route C-F (80), and this new route is assigned an occurrence of **80**. The remaining (unreplaced) route, A-B-C, is assigned an occurrence of 100–80=20. The modified routes list is thus:

A-B-C (20)
B-D (200)
A-B (250)
A-B-C-F (80),

for which the sum of the occurrences is **550**. As will be recognized by comparing this modified routes list to the original routes list, this process of merging consecutive routes does not affect the load distribution associated with the routes list.

The above-described process of merging consecutive routes is repeated iteratively until either the sum of the occurrences does not exceed the number of Vusers, or no consecutive routes remain in the routes list. In the present example, the second iteration of this process results in the merger of routes A-B (250) and B-D (200), resulting in the following routes list:

A-B-C (20)
A-B-D (200)
A-B (50)
A-B-C-F (80).

At this point, no consecutive routes remain in the list. Thus, the process proceeds to phase 2.

The general function of phase 2 is to condense the routes list into a shorter list of longer routes (while maintaining general congruence between the lengths of the routes), and then to assign the resulting routes to Vusers (or Vuser groups) as Web scripts. Some modification to the original load distribution typically occurs as the result of this process.

With reference to block **350** of FIG. **32**, the first step of this phase 2 process involves sorting the routes list according to the occurrence values. This produces the following list:

A-B-C (20)
A-B (50)
A-B-C-F (80)
A-B-D (200).

With reference to decision block **352**, if the sum of the occurrences is greater than the number of Vusers, the process enters into a loop (blocks **352–360**), and remains in this loop as long as the condition is met. The first processing step of this loop, which is represented by block **354**, involves locating the pair of adjacent routes for which the sum of route lengths is smallest (wherein route length refers to the number of URLs in the route), and then concatenating the two routes of the pair to form a single route. This new route replaces the concatenated routes, and is assigned an occurrence equal to the average of the occurrences of the concatenated routes. In the present example, the first iteration of this step produces a concatenation of routes A-B-C (20) and A-B (50), resulting in the following list:

A-B-C-A-B (35)
A-B-C-F (80)
A-B-D (200).

The second step of this loop, represented by block **360**, involves dividing all of the occurrences by the smallest

occurrence value (unless the smallest occurrence value is one or lower). The division in this step and step **352** may be either regular division (as in this example) or integer division. The result of this step produces the following list:

A-B-C-A-B (1)

A-B-C-F (2.29)

A-B-D (5.71),

which has a sum of occurrences of 9. Because 9 is less than 10 (the number of Vusers), the process exits the program loop at step **352**.

With reference to block **364**, the occurrence values are then adjusted, if necessary, to produce a set of integer values for which the sum is equal to the number of Vusers. Any scaling/truncating technique that maintains the general proportions between the occurrence values may be used for this purpose. This step may, for example, produce the following:

A-B-C-A-B (1)

A-B-C-F (3)

A-B-D (6).

With reference to blocks **366** and **368**, each route is then outputted as a respective Web script file. In addition, the process generates a scenario file that specifies the allocations of Vusers to Web scripts. In this example, the scenario would include a single Vuser that runs the A-B-C-A-B script, a group (Sgroup) of three Vusers that run the A-B-C-F script, and a group of six Vusers that run the A-B-D script. The scenario file would also include the loop information specified by the user.

(d) Source Code Listing

Included as Appendix C is a source code listing which includes an implementation of the two-phase process. The listing also includes code for implementing the Vusers.

XII. Conclusion

While certain preferred embodiments of the invention have been described, these embodiments have been presented by way of example only, and are not intended to limit the scope of the present invention. For example, although the present invention has been described with reference to the standard protocols, services and components of the World Wide Web, it should be recognized that the invention is not so limited, and that the various aspects of the invention can be readily applied to other types of web sites and server applications (including, for example, mainframe terminal applications). Accordingly, the breadth and scope of the present invention should be defined only in accordance with the following claims and their equivalents.

In the claims which follow, reference characters used to designate claim steps are provided for convenience of description only, and are not intended to imply any particular order for performing the steps.

What is claimed is:

1. A computer-implemented method of generating scripts that are adapted to be played to exercise a web site, comprising:

processing a server access log associated with the web site to identify a plurality of navigation routes followed by visitors of the web site during ordinary, post-deployment usage of the web site, the server access log reflecting actions of a plurality of said visitors; and

translating the plurality of navigation routes into at least one script that specifies a sequence of client request messages for exercising the web site.

2. The method as in claim **1**, wherein translating the plurality of navigation routes into at least one script comprises preserving a general distribution of access requests among web site content entities as reflected within the server access log.

3. The method as in claim **1**, wherein processing the server access log to identify a plurality of navigation routes comprises identifying first and second content entities accessed by a visitor in sequence, and determining whether the web site includes a navigational link between the first and second content entities.

4. The method as in claim **3**, further comprising incrementing a counter associated with the navigational link to indicate that the visitor followed the navigational link.

5. The method as in claim **1**, wherein processing the server access log comprises determining of a number of times each of a plurality of navigational links of the web site was followed by a visitor.

6. The method as in claim **1**, wherein translating the plurality of navigation routes into at least one script comprises merging consecutive routes of the plurality of navigation routes.

7. The method as in claim **1**, wherein the script comprises a plurality of Uniform Resource Locators associated with the web site.

8. The method as in claim **1**, wherein the method comprises translating the plurality of navigation routes into a scenario that comprises multiple scripts.

9. The method as in claim **8**, wherein translating the plurality of navigation routes into a scenario comprises preserving a general distribution of access requests among web site content entities as reflected within the server access log.

10. The method as in claim **1**, wherein the server access log comprises a standard-format server access log file.

11. The method as in claim **10**, wherein the standard-format server access log file comprises an HTTP Server Access Log file or an HTTP Server Referrer Log file.

12. The method as in claim **1**, further comprising running the at least one script while monitoring server response times to client request messages specified therein.

13. A computer-readable medium having stored thereon a computer program which, when executed by a computer:

processes a server access log reflective of actions of a plurality of visitors of a web site to identify a plurality of navigation routes followed by said visitors of the web site; and

translates the plurality of navigation routes into at least one script that specifies a sequence of client request messages for exercising the web site.

14. The computer-readable medium as in claim **13**, wherein the computer program translates the plurality of navigation routes into at least one script so as to preserve a general distribution of access requests among web site content entities as reflected within the server access log.

15. The computer-readable medium as in claim **13**, wherein the computer program processes the server access log to identify a plurality of navigation routes by at least identifying first and second content entities accessed by a visitor in sequence, and determining whether the web site includes a navigational link between the first and second content entities.

16. The computer-readable medium as in claim **15**, wherein the computer program increments a counter associated with the navigational link to indicate that the visitor followed the navigational link.

17. The computer-readable medium as in claim **13**, wherein the computer program uses the server access log to determine a number of times each of a plurality of navigational links of the web site was followed by a visitor.

18. The computer-readable medium as in claim 13, wherein the computer program translates the plurality of navigation routes into at least one script by at least merging consecutive routes of the plurality of navigation routes.

19. The computer-readable medium as in claim 13, wherein the script comprises a plurality of Uniform Resource Locators associated with the web site.

20. The computer-readable medium as in claim, 13, wherein the computer program translates the plurality of navigation routes into a scenario that comprises multiple scripts.

21. The computer-readable medium as in claim 20, wherein the computer program translates the plurality of navigation routes into a scenario so as to preserve a general distribution of access requests among web site content entities as reflected within the server access log.

22. The computer-readable medium as in claim 13, wherein the server access log comprises a standard-format server access log file.

23. The computer-readable medium as in claim 22, wherein the standard-format server access log file comprises an HTTP Server Access Log file or an HTTP Server Referrer Log file.

24. The computer-readable medium as in claim 13, further comprising a computer program which runs the at least one script while monitoring server response times to client request messages specified therein.

25. A computer-implemented method of evaluating the performance of a web site, comprising:

processing a server access log associated with the web site to generate a plurality of scripts such that a general distribution of access requests among web site content entities as reflected within the server access log is preserved; and

executing the plurality of scripts to exercise the web site while monitoring web site performance.

26. The method as in claim 25, wherein processing the server access log comprises tracing navigation paths followed by visitors during ordinary, post-deployment usage of the web site to identify a plurality of navigation routes followed by the visitors, and incorporating the plurality of navigation routes into the plurality of scripts.

27. The method as in claim 25, wherein processing the server access log comprises determining of a number of times each of a plurality of navigational links of the web site was followed by a visitor.

28. The method as in claim 25, wherein each script comprises a plurality of Uniform Resource Locators associated with the web site.

29. The method as in claim 25, wherein the server access log comprises a standard-format server access log file.

30. The method as in claim 29, wherein the standard-format server access log file comprises an HTTP Server Access Log file or an HTTP Server Referrer Log file.

31. The method as in claim 25, wherein monitoring web site performance comprises monitoring response times to client requests.

32. The method as in claim 25, wherein executing the plurality of scripts comprises load-testing the web site.

* * * * *